DESKTOP REFERENCE

SE
SHOEISHA

C/C++
辞典 第3版

関数・シンタックス・アルゴリズム
逆引きリファレンス

Written by Shunji HYUGA

C言語とC++の
「そこが知りたい」が
すぐわかる!

これからもC/C++を使い続ける人のために
パワーアップして新登場!

組み込みの
開発現場でも
大活躍!

DESKTOP REFERENCE

SE
SHOEISHA

C/C++
辞典 第3版

関数・シンタックス・アルゴリズム
逆引きリファレンス

Written by Shunji HYUGA

本書内容に関するお問い合わせについて

このたびは翔泳社の書籍をお買い上げいただき、誠にありがとうございます。弊社では、読者の皆様からのお問い合わせに適切に対応させていただくため、以下のガイドラインへのご協力をお願い致しております。下記項目をお読みいただき、手順に従ってお問い合わせください。

● ご質問される前に

弊社Webサイトの「正誤表」や「出版物Q&A」をご確認ください。これまでに判明した正誤や追加情報、過去のお問い合わせへの回答（FAQ）、的確なお問い合わせ方法などが掲載されています。

正誤表　　http://www.seshop.com/book/errata/
出版物Q&A　http://www.seshop.com/book/qa/

● ご質問方法

弊社Webサイトの書籍専用質問フォーム（http://www.seshop.com/book/qa/）をご利用ください（お電話や電子メールによるお問い合わせについては、原則としてお受けしておりません）。

※ 質問専用シートのお取り寄せについて

Webサイトにアクセスする手段をお持ちでない方は、ご氏名、ご送付先（ご住所／郵便番号／電話番号またはFAX番号／電子メールアドレス）および「質問専用シート送付希望」と明記のうえ、電子メール（qaform@shoeisha.com）、FAX、郵便（80円切手をご同封願います）のいずれかにて“編集部読者サポート係”までお申し込みください。お申し込みの手段によって、折り返し質問シートをお送りいたします。シートに必要事項を漏れなく記入し、“編集部読者サポート係”までFAXまたは郵便にてご返送ください。

● 回答について

回答は、ご質問いただいた手段によってご返事申し上げます。ご質問の内容によっては、回答に数日ないしはそれ以上の期間を要する場合があります。

● ご質問に際してのご注意

本書の対象を越えるもの、記述個所を特定されないもの、また読者固有の環境に起因するご質問等にはお答えできませんので、あらかじめご了承ください。

● 郵便物送付先およびFAX番号

送付先住所　〒160-0006　東京都新宿区舟町5
FAX番号　03-5362-3818
宛先　（株）翔泳社 編集部読者サポート係

※ 本書の内容については正確な記述に努めましたが、著者および出版社は、本書の内容およびサンプルプログラムに対してなんら保証をするものではなく、また本書の内容およびサンプルプログラムによるいかなる運用結果についても、一切の責任を負いません。またC/C++プログラミングに関する、本書の記述を超える内容や、特定の条件（特定のコンパイラ実装、特定の開発・実行環境など）に関連するご質問については、著者および出版社ではお答えいたしかねる場合があります。

はじめに

C言語とC++（以下、C/C++）は、現在のプログラミングにおいて最も重要で強力なプログラミング言語のひとつです。現在、広く利用されている主要なOSやウィンドウシステムとデバイスドライバの中には、その一部または全体がC/C++で開発されているものがあります。また、多くの優れたアプリケーションプログラムやコンポーネントの中にも、C/C++で開発されているものがたくさんあります。組み込みシステムの開発にもC/C++が使われるようになってきました。つまり、OSやウィンドウシステムを利用するアプリケーションを開発するときから、デバイスの低レベルの制御まで、C/C++の守備範囲は広範です。また、ソースファイルで提供されている既存のプログラムやデバイスドライバをインストールしたりOSのカーネルを再構築したりするようなシステム管理の作業を行うためにも、C/C++の知識は不可欠です。

ところで、実際のプログラミングの現場では、C言語あるいはC++のいずれか一方の知識だけでは不十分であることがよくあります。C++はC言語を拡張して作成された言語であるため、C言語の知識にC++の知識を追加し、オブジェクト指向の考え方を理解すれば、C言語とC++の両方を自在に使いこなせるようになります。たとえば、C言語とC++のシンタックスの大半は同じですし、C++のプログラムではC言語の関数をほとんどそのまま使うことができます。C言語とC++は別のものと考えるのではなく、同じファミリーの兄弟として、どちらも理解しておく必要があるでしょう。

本書は、C言語とC++のシンタックスやライブラリなど、プログラミングの際に必要となることがらを、適切なサンプルと共に簡潔にまとめたものです。本書の第1版と第2版はたくさんの読者の皆様に愛用されました。C/C++は標準化が進んでいるので、以前の版の執筆時と状況が大きく変わったわけではありませんが、この第3版では現状にあわせていくつかの増補を行い、同時に若干の不適切な部分を見直しました。C言語やC++を使うときには、本書をつねに手元において、必要なときにいつでも活用してください。

最後になりますが、本書刊行にあたってご尽力いただいた、株式会社翔泳社編集部をはじめ多くの皆様に感謝します。

日向俊二

本書の内容

本書の内容は、International Standards Organization（ISO）/American National Standards Institute（ANSI）標準に加えて、各種C/C++処理系のドキュメントなどの情報をもとに、現在最も一般的で普遍的なC/C++プログラムを記述する際に役立つ情報をまとめたものです。本書は標準の解説書ではありませんので、ISO/ANSIやPOSIXなどの標準をすべて解説しているわけでも、すべての点で最新の標準に適合していることを保証するわけでもありません。また、標準および処理系のトピックで、あまり使われていないことがらは省略しています。必要な場合はそれぞれの標準のドキュメントを参照してください。

01 C/C++言語の基礎知識

C言語とC++プログラムの構造、プログラム開発の手順やコンパイルの過程、C++プログラミングおよびプログラミング言語としてのC/C++とシステムやライブラリとの関連の概要を説明します。ここで説明することは本書のリファレンス部分を読む際に必要な基礎知識です。

02 C/C++のシンタックス

プログラミング言語としてのC++とC言語の基礎を説明します。C++の言語構造や規則などの基礎のほかに、例外処理などについても解説します。Cのライブラリ関数とC++のクラスを除く、C/C++の基本的な要素のほとんどのことはこの章を参照するとわかります。

03 C言語リファレンス

C言語の標準ライブラリに含まれる関数を、関数の種類ごとに説明します。このリファレンスの全体に目を通すと、C言語ライブラリ関数を使ってできることを把握することができます。また、目的に応じて必要な部分だけを参照してもかまいません。

04 C++リファレンス

標準C++ライブラリに含まれるプログラミング要素のリファレンスです。C++のシンタックスはC言語によく似ているので「02 C/C++のシンタックス」で説明しています。

05 そのほかのライブラリ

ウィンドウシステム、OpenGLなど、現代のC/C++のプログラミングで特に重要なライブラリについて概説します。

付録A C/C++プログラミングのファイル

C/C++プログラミングに関連するファイルを、コマンド、ファイル、標準的なファイル拡張子に分けて説明します。

付録B トラブル対策

一般的なトラブルやコンパイル・実行時のエラーメッセージとその対処などを示します。

付録C プログラミング用語集

C/C++プログラミングで重要な用語と、特有の意味を持つ用語を解説します。

付録D 参考リソース

参考文献およびWebサイトを示します。

本書の表記について

記号と表記

本書では、以下のような記号や表記を用います。

記号	意味
C++	C++固有のことであることを示します。このマークを付けて説明していることは、C言語には当てはまりません。
参照→	参照する項目やページを示します。
xxx()	printf()やscanf()などの関数、およびC++の各クラスのメソッドの名前には()を付けて表記します。xxx()関数または関数xxx()と明記していない場合でも、最後に()を付けてあるシンボルは原則として関数名です。
[ ]	書式で、省略可能なパラメータを表します。たとえば、 for([初期化式]; [条件式]; [ループ式]) の場合、初期化式、条件式、ループ式のいずれでも省略できます。
バックスラッシュ	（\）はWindowsのような普及している日本語環境では通常は円記号（¥）として表示されます。そのため、本書では円記号（¥）を使います。

本書の用語

本書でよく使われる用語の定義は以下のとおりです。

用語	意味
変数	値を代入できるあらゆる変数を指します。構造体変数、オブジェクト変数などを含みます。
式	式には、通常の計算式のほか、論理式と、1つの値が含まれます。たとえば、if (式)という場合、式は値でもかまわないので、if (n)のように式の内容が変数1つだけでもかまいません。
ステートメントブロック	ステートメント1つ、あるいは複数のステートメントを表します。C/C++では{ }の中に一連のステートメントを記述できますが、このようなところに記述するステートメントブロックを省略して「ステートメント」と表記することがあります。

本書の中で「慣例」と明示している部分は、必ずしも従う必要はないものの、覚えておいて、必要に応じて利用するとよいことです。

関数・クラスの説明

関数・クラスのリファレンスでは、以下のような項目別に説明します。

項目	意味
ヘッダ	その関数またはクラスを使うときに必要な標準的なファイルを示します。実際に必要なヘッダファイルは処理系によって異なることがあります。
書式	書式を示します。
引数	引数（パラメータ）を説明します。書式がfunc(void)になる関数は引数がない関数です。引数がない関数の場合、「引数」は省略します。
戻り値	関数が返す値を示します。書式がvoid func(...)になる関数は戻り値がない関数です。戻り値がない関数の場合、「戻り値」は省略します。
解説	この項目に関する解説です。
注意	特に注意を払いたいことを説明します。
参照	重要な関連がある項目や解説を補足する項目がある場合にその項目を示します。
エラーコード	エラーが発生したときにセットされるエラーコードは、システムによって異なることがあります。代表的なコードのシンボルを示します。
準拠	準拠している仕様または標準を示します。
互換性	環境依存することがらを示します。
例	プログラム例を示します。
関連項目	関連する項目の見出しを示します。

本書の表記について

リファレンスの章の主な略語

リファレンスの章で書式を示す際には、以下のような略語を使います。

略語	意味
addr	アドレス（アドレスを保存するバッファまたはsockaddr構造体などのポインタ）
addrlen	addrのサイズ、または、アドレスの長さを保存するポインタ
amode	アクセス許可のフラグの組み合わせ
arg	引数、または引数リスト
arg_ptr	引数リストのポインタ
array	配列の名前
buf	バッファ
bufn	bufと同様のバッファ（nは1,2などで、buf1、buf2などの形式で複数のバッファを識別）
cast_expr	キャスト式
class	クラス名
compar	比較関数名
cond_expr	条件式
const_expr	定数式
decl	宣言
decl_list	宣言リスト
dir	ディレクトリ
except	例外
except_decl	例外の宣言
expr	式（変数、ポインタ変数、ポインタを返す関数名などを含む）
exprn	exprと同様の式（nは1,2,3などで、expr1、expr2、expr3などの形式で複数の式を識別）
fd	ファイルのデスクリプタ
fname	ファイル名

略語	意味
fp	ファイルポインタ
Fun	関数（Function）
id	識別子
iiter	入力反復子（InputIterator）
init_expr	初期化式
iter	反復子（Iterator）
loop_expr	ループ式
member	クラスや構造体のメンバー
member_func	メンバー関数
member_list	構造体や共用体のメンバーのリスト
member_name	メンバー名
mnt	mntent構造体のポインタ
mode	ファイルまたはデバイスのモード
path	ファイル名またはパス名
ptr	ポインタ
statement	ステートメントまたはステートメントブロック
statn	statementと同様の式（nは1,2,3などで、stat1、stat2、stat3などの形式で複数のステートメントを識別）
str	文字列
str_literal	文字列リテラル
stream	ファイルまたは入出力ストリーム
struct_def	構造体定義
tag	構造体、共用体、列挙型などを識別するタグ
type	型（変換する型、型情報を取得する型など）
var	変数

CONTENTS

CONTENTS

CONTENTS

CONTENTS

01：C/C++言語の基礎知識

ここでは、C言語とC++プログラムの構造、プログラム開発の手順やコンパイルの過程、C++プログラミングおよびプログラミング言語としてのC/C++とシステムやライブラリとの関連の概要を説明します。ここで説明することは本書のリファレンス部分を読む際に必要な基礎知識です。

01 C言語とC++言語

C言語は、現在、最も重要で広く使われているプログラミング言語の1つです。C++はC言語を拡張して作成されたオブジェクト指向プログラミング言語です。

C言語とC++のシンタックスの大半は同じなので、C++のプログラムではC言語のコードをほとんどそのまま使うことができます。そのため、C++固有のいくつかの点を除くと、C言語とC++はほとんど同じものと考えることも可能です。しかし、それぞれの言語を適切に利用するためには、それぞれの言語の特性をきちんと把握しておく必要があります。

C言語

プログラミング言語Cは、さまざまな目的に使われている、最も重要なプログラミング言語の1つです。また、C言語は、現在、いろいろな分野で広く使われている多様なプログラミング言語の中では、歴史が比較的長く、標準化が進んでいるため、さまざまなシステムでコンパイルして実行できるプログラミング言語でもあります。

C言語の特徴は次のとおりです。

- C言語はプログラミング言語として標準化されており、そのライブラリもまた標準化されています。システムコールやウィンドウシステムのようなプラットフォーム固有の機能に依存する部分を除いて、プログラムの移植が比較的容易です。
- システムのリソース（メモリやCPUなど）を直接利用する、OSやデバイスドライバのような低レベルのプログラムの記述にも適しています。組み込み機器のソフトウェアのプログラミングにもよく使われます。
- コンパクトで高速なプログラムを記述したり生成したりすることが可能です。
- C言語で記述したモジュールは、他のプログラミング言語とのリンクの標準的な方法として使われることが多く、さまざまなプログラミング言語のプログラムとのリンクが比較的容易です。

▼ リスト1.1　C言語の最も基本的なプログラムの例

```
/*
 * hello.c
 */

#include <stdio.h>

int main(int argc, char* argv[])
{
   printf("Hello C/C++¥n");

   return 0;
}
```

C++

プログラミング言語C++は、オブジェクト指向プログラミングのために生まれたプログラミング言語です。

C++はオブジェクト指向プログラミングのための言語ですが、従来のC言語のプログラムにC++の既成のクラスを活用して、オブジェクト指向ではない、手続き型のプログラミングを行うこともできます。そのため、C++は完全なオブジェクト指向プログラミング言語というより、オブジェクト指向プログラミングが可能な複合型の言語といえます。

C++言語のオブジェクト指向プログラミングでは、クラスを定義して、クラスから作成したオブジェクトを利用します。クラスにはデータと操作や処理を行うためのコード（メンバ関数）が含まれます。

C++の特徴は次のとおりです。

- 比較的複雑で大規模なプログラムの、オブジェクト指向プログラミングに適しています。
- C++のプログラムの中にC言語の関数を記述したり、C言語で記述したモジュールと容易にリンクすることができます。
- C++のシンボルにはコンパイラが適宜文字を追加する（名前の装飾）ので、ほかのプログラミング言語からC++の関数をそのまま呼び出すことは、通常、できません。

01-01 C言語とC++言語

・C言語に比べるとC++のプログラムの方が実行可能コードのサイズが大きくなり、実行時のパフォーマンスも劣る傾向があります。しかし、一般にインタープリタ言語に比べるとかなり高速です。

▼ リスト1.2 C++の最も基本的なプログラムの例

```
//
// Simple.cpp
//

#include <iostream>

using namespace std;

// Moneyクラスの宣言
class Money {
public:
   Money() { balance = 0; }; // コンストラクタ
   ~Money() {}; // デストラクタ
   void setYenRate(int Rate);
   void setDollar(double doll);
   int getYen() { return balance; };
private:
   int balance; // 現在の残高(円単位)
   int YenRate; // 1$に対する円
};

// Moneyクラスの実装
void Money::setYenRate(int Rate)
{
   YenRate = Rate;
}

void Money::setDollar(double doll)
{
   balance = (int)(YenRate * doll);
}

// Moneyクラスを利用するmain()
int main(int argc, char* argv[])
{
   int Rate;
   double value;
```

```
    Money* dv = new Money();
    cout << "1$ は円でいくら?>";
    cin >> Rate;

    dv->setYenRate(Rate);
    cout << "ドル($)の価格は>";
    cin >> value;
    dv->setDollar(value);
    cout << "日本円の価格= " << dv->getYen() << endl;

    return 0;
}
```

プログラミング豆知識

標準の意味

標準化の重要性は、プログラミングの対象とプログラムが複雑になればなるほど増しています。一度書いたプログラムコードを、できるだけ広い範囲で長く使えるようにするためには、可能な限り標準に従っておくことが重要です。

しかし、標準といっても、それ自身、完全であるわけでも、永久不変であるわけでもありません。標準には完全には定義されてない要素もあります。たとえば、ANSI Cのintのサイズは最小のサイズが決められているだけで、実際のサイズは実装(コンパイラ)で決まることになっています。また、標準に含まれない拡張された書式や関数などで、ある状況でとても便利であったり、特定の目的のために特に必要になることもあります。そのため、現実のC/C++コンパイラはほとんどすべて、標準から拡張されたC/C++言語で書かれたソースコードをコンパイルすることができます。また、多くのコンパイラは、オプションを指定することで、受け入れる言語拡張の種類を選択できるようになっています。

一般的にいって、標準と非標準で選択が可能な場合には、標準に従うことが望ましいでしょう。特に移植性を重視する場合、長期間使うことを想定しているプログラムの場合、特定の試験の受験などでは、標準に準拠することが重要になる場合があります。しかし、既存のC/C++プログラムの資産には、標準に準拠しないソースコードもたくさんあるので、標準以外の書き方や使い方があることも知っておく必要があります。

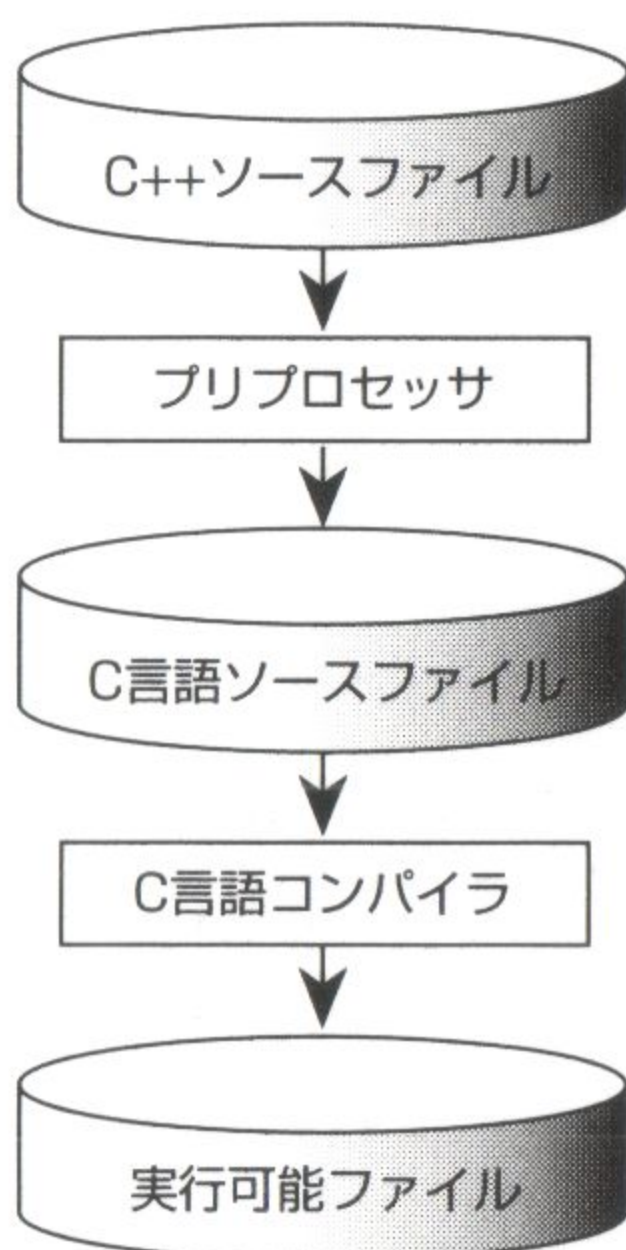
02 C言語とC++の関係

C++が誕生したころは、C++はC言語の一種の拡張と考えられていました。C++が生まれ育った経緯と、C++とC言語との関係を理解しておくと、C++のプログラミングで役に立ちます。ここでは、C++とC言語の関係について概説します。

C++の成立過程

C++は、現在のプログラミングにおいて最も重要で強力な言語です。しかし、C++はオブジェクト指向のプログラミング言語としてゼロから作成されたプログラミング言語ではありません（最初からオブジェクト指向言語として作られたプログラミング言語には、たとえばC#やJava、Smalltalkなどがあります）。

初期のC++はC言語のプリプロセッサ（前処理プログラム）を使って処理する言語として作られました。

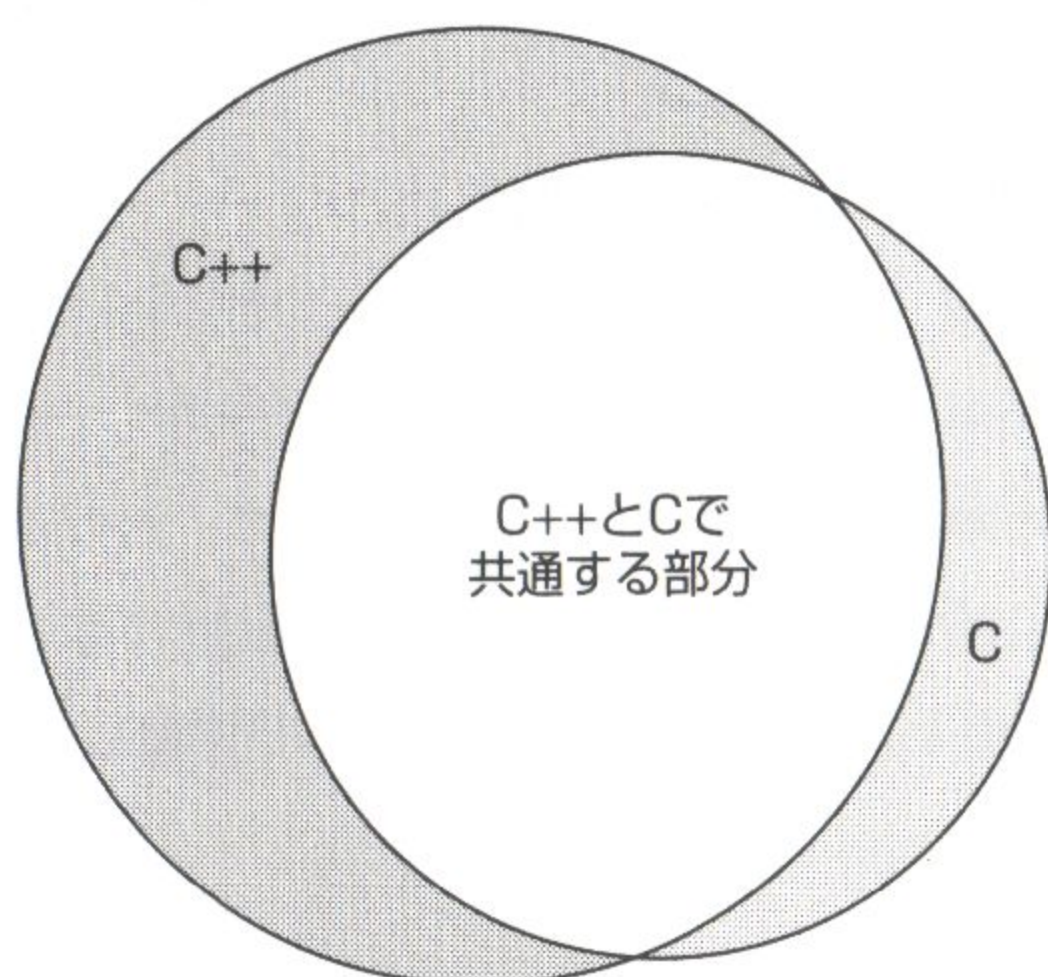
C++ソースファイル
↓
プリプロセッサ
↓
C言語ソースファイル
↓
C言語コンパイラ
↓
実行可能ファイル

● 図1.1　初期のC++処理系

そして、現在でもこの考え方は基本的な部分で維持されています。そのため、C++では次のようにC言語の要素を利用することができます。

・C言語のキーワード（ステートメントやディレクティブなど）は、C++でもほとんど同じように使うことができます（C++にはC言語のキーワードにC++固有のキーワードが追加されています）。

・ほとんどの場合、C++ではC言語の関数をそのまま利用できます。

・クラスに属さないもの（変数、定数、関数など）を定義して使うことができます（JavaやC#のような完全なオブジェクト指向プログラミング言語では、すべてがいずれかのクラスに属します）。

・普通、C++のコンパイラはC言語のプログラムもコンパイルできます。しかし、逆に、C++固有の要素は、通常、C言語では直接利用できません。つまり、C++のプログラムの中にC言語のコードを挿入してC++のプログラムとしてコンパイルすることはできますが、C言語のソースプログラムの中にC++のコードを混ぜると、C言語としてコンパイルできません。

● 図1.2　C言語とC++の関係

このようなC++の性質は、オブジェクト指向プログラミング言語としては特別な性質といえます。つまり、非オブジェクト指向のC言語の関数を呼び出すことができたり、C++のソースの一部にC言語の形式でプログラムを記述することが可能であるだけでなく、C++でオブジェクトをまったく使わない手続き型のC言語的なプログラミングさえ可能です。

C言語とC++のプログラミングスタイル

C++とC言語の言語キーワードはほとんど同じです。また、C++のプログラムの中ではC言語のコードをほとんどそのまま使うことができます。しかし、C++のプログラミングの方法は、本来、C言語のプログラミングの方法とは異なります。

C言語のプログラミングスタイル

端的に言えば、C言語のプログラミングとは関数を作ることです。

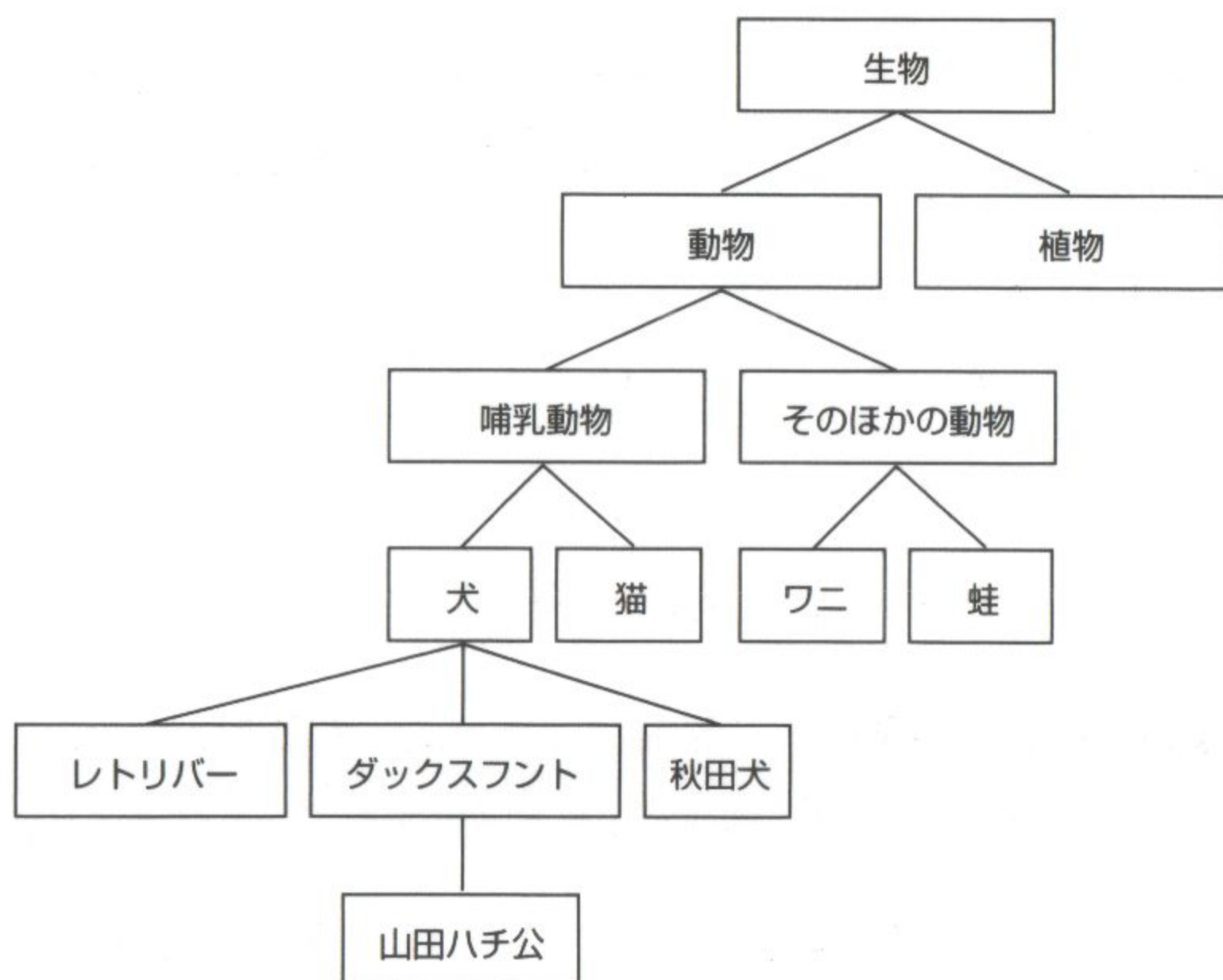
C言語のプログラミングでは、目的を達成するための機能を分割して、それぞれ独立し

た関数として記述します。たとえば、単純な住所録アプリケーションを作る場合、ユーザーからの入力を受け取るための関数、入力データに矛盾がないかどうか調べる関数、データをファイルに保存する関数、ファイルからデータを探して取り出す関数、特定の住所録レコードを表示する関数、住所録のリストを表示する関数など、さまざまな関数を記述し、それぞれの関数を適切な機会に呼び出すようにプログラムします。

C++のプログラミングスタイル

端的に言えば、C++のプログラミングとはクラスを定義して利用することです。

C++は、オブジェクト指向プログラミングのためのプログラミング言語です。オブジェクト指向のアプローチでは、一連のものの共通する特性を突き止めて分類し（抽象化）、クラスを定義します。たとえば、住所録アプリケーションを作る場合、住所と氏名などからなる個人の情報のクラスと、そのクラスのインスタンス（オブジェクト）に氏名や住所などを保存するためのC++のクラスのメンバ関数（メソッド）を記述します。

C++ではものごとを抽象化してオブジェクトとしてとらえますが、このような作業は特別なことではありません。日常生活の中でも、我々は意識せずにものの特性を見極めて分類し、クラスとしてモデル化します。たとえば、近所の野良犬から山田さんの家のハチ公まで、犬とみなせる動物をすべて犬として扱って、ほかの動物と区別します。そして、犬は哺乳動物に属するということを知ると、犬を、他の哺乳動物が持つ特性と同じ特性を持つものとして認識します。

● 図1.3 オブジェクト指向のアプローチ

プログラミングにおけるオブジェクト指向のアプローチは、このような考え方をプログラミングに導入したものととらえることができます。

オブジェクト指向のプログラミングでは、あらゆることを想定し、よく分析し検討して可能な限り現実の事象を論理的に矛盾なく扱いやすいように抽象化してモデルを考えま

す。オブジェクトのモデル化の方法や結果は、目的や対象によって異なります。また、最初の段階ですべてを完全にとらえることが事実上不可能なため、試行錯誤を伴う作業になることがあるのがオブジェクト指向プログラミングの特徴です。たとえば、図1.3では動物を哺乳動物とそれ以外の動物に分けました。実生活においてはこの程度の分類で十分な場合が多いでしょうが、動物を学術的に扱うときには、哺乳類、甲殻類、両棲類などのように類で分ける必要があるでしょう。また、目的によっては、「ワンと吠える」、「ニャオと鳴く」、「吠えも鳴きもしない」という基準で分類するほうが合理的である場合もあります。分類のしかたは目的によって異なり、いずれが正解でどれが間違いということはありません。プログラミングでも同じことで、目的に合わせて最適なモデル作りを目指します。

C言語とC++の主な違い

C言語とC++の主な違いを、以下に示します。

- C++にはnewとdeleteやusingのようなC言語にはないキーワードがあります。コンパイラによってはC言語でもC++のキーワードが予約されている場合があります(C言語のプログラムの中でC++のキーワードを変数名などの識別子として使えない場合があります)。
- C++では、関数の引数にデフォルト値を指定できます。
- C++ではシグネチャ（引数の数や型、戻り値の型）が異なる複数の同じ名前の関数を宣言して使うこと（オーバーロード）が可能です。C言語では同じ名前の関数を作ることはできません。
- C++では、インライン関数や、仮想関数がサポートされています。
- C++では、テンプレートがサポートされています。
- C++では、例外が言語でサポートされています。C言語ではコンパイラがサポートします。
- C++ではtypedef宣言しないで構造体を型として宣言できます。C言語ではtypedefを使って構造体にエイリアスを定義する必要があります。
- C++では、ステートメントブロックの中でローカルなオブジェクト宣言を行うことができます。たとえば、for (int i=0;;)のようにforステートメントの中でもローカルな変数を宣言できます。
- ANSI C++では、関数をネストできます。
- C++では、すべての関数は正式のプロトタイプを宣言する必要があります。また、C++では、型チェックはC言語より厳しく行われます。

01 C言語プログラムの構造

作成するプログラムの種類に関係なく、C言語とC++のソースプログラムの基本的な構造は同じです。

C++がC言語と異なることは、C言語の構成要素に加えてオブジェクト指向プログラミングに必要な要素がC++には追加されるという点です。

最も一般的なC言語プログラムの構造はリスト1.3に示すとおりで、次のような構成になります。

▼リスト1.3　一般的なC言語プログラムの構造

```
/*
 * normalw.c - 標準体重を計算するプログラム
 */
#include <stdio.h>
#include <locale.h>
#define BMI 22

double CalcSW(int height);

int main(int argc, char *argv[])
{
   int height;
   setlocale( LC_ALL, "Japanese" );
   //ユーザーの入力
   printf("身長(cm)を入力してください>");
   scanf("%d" , &height);
   //標準体重を表示する
   printf("標準体重%.2f kg¥n" , CalcSW(height));

      return 0;
}

//標準体重を計算する
double CalcSW(int height)
{
   double h;
   h = height * 0.01;
   return (h * h* BMI);
}
```

ヘッダコメント
#includeディレクティブ
定義と宣言
main()
ユーザー定義の関数

ヘッダコメント

プログラムの名前や簡単な説明などを記述したコメントです。このコメントはなくてもかまいませんが、学習や実験用のきわめて小さなプログラムを除いて、どのソースプログラムファイルでも先頭にそのファイルの内容を示すコメントを記述するべきです。

#includeディレクティブ

ほかのファイルを、そのソースプログラムファイルのその場所にインクルードする（取り込む、挿入する）ためのディレクティブ（一種の命令）です。ファイルの先頭（ヘッダコメントのあと）に、そのプログラムで使っている関数や定数を宣言しているモジュールのヘッダファイルを、#includeを使ってインクルードします。

C言語やC++では、あらかじめ宣言されていないシンボルはintとみなされるか、あるいは使えないので、そのプログラムで使っているシンボルを定義しているヘッダファイルを必ずインクルードしなければなりません。

定義と宣言

通常、そのファイル全体に有効な定数の定義や、関数のプロトタイプ宣言は、実行されるコードより前に記述します。

関数のプロトタイプ宣言とは、関数の引数の型と関数名（識別子）を宣言することです。自分で作った関数を、その定義より前に参照している場合、C言語でもプロトタイプを宣言する必要がある場合があります。たとえば、リスト1.3の場合、関数main()で関数CalcSW()を呼び出していますが、CalcSW()の定義はmain()のあとにあるので、プロトタイプdouble CalcSW (int height);を宣言しています。関数のプロトタイプ宣言はヘッダファイルに記述することもできます。

main()

C言語のプログラムは、プログラムが起動すると自動的に実行される特別な関数であるエントリー関数から実行されます。エントリー関数の名前は通常はmain()です。

main()に代わる特定のエントリー関数が用意されている場合もあります。Win32コンソールアプリケーションでよく使われる_tmain()、WindowsのGUIアプリケーションをC言語で記述するときに使われてきたWinMain()や、Unicodeプログラミングモデルに準拠するプログラムを記述するときに使われてきたUnicode用のwmain()などを、main()の代わりに使うことがあります。

プログラムのメインモジュールには、main()かそれに代わる名前の関数が必要です。main()の中では、式を記述したり、C言語であらかじめ提供されている関数（→「03 C言語リファレンス」）や自分で作成した関数を呼び出してプログラムに必要な操作や処理を

行います。また、プログラムのロジックの一部あるいは大部分を自分で作成した別の関数（ユーザー定義の関数）に記述して、main()からその関数を呼び出すこともできます。C++のプログラムの場合は、main()の中でオブジェクトを作って、クラスのメンバ関数（→「04 C++リファレンス」）を呼び出すことができます。

関数と宣言

プログラマが自分で作成した関数は、プロトタイプを宣言したら、main()の前後のいずれに置いても、あるいはほかのソースファイル（モジュール）に置いてもかまいません。また、必要であれば、関数と関数の間にグローバルな変数を記述することもできます（たとえば、main() {...}のあとでCalcSW()の前に、それ以降に記述した関数で使う変数を宣言できます）。

02 C++プログラムの構造

一般的なC++のプログラムは、次のような3種類の要素から成り立っています。実際のプログラム開発では、リスト1.4とリスト1.5に示すように、種類ごとに個別のファイルに記述するのが普通です。

・クラスの宣言やインターフェイスの定義

C++のクラスの宣言やインターフェイスは、通常はそのクラスのヘッダファイルに記述します。（リスト1.5）

・クラスの実装

クラスのメンバ関数は、通常はそのクラスの実装（インプリメンテーション）ファイルに記述します。（リスト1.4）

・クラスを利用するプログラム本体

クラスを利用するプログラムは、C言語のプログラムと同じようにエントリー関数main()からプログラムが実行されます。（リスト1.4）

C++のプログラムは当初はC言語から拡張された言語なので、クラスを使わないで、C言語のプログラムと同じように関数呼び出しだけで構成されるプログラムとして記述することも可能です。また、標準C++ライブラリに含まれるクラスやそのほかの既存のクラスだけを利用して、クラスの実装を記述しないプログラムとして記述することも可能です。

▼ リスト1.4　一般的なC++プログラムの構造（実装ファイル）

```
//
// main.cxx                                  ヘッダコメント
//
#include <iostream>                          #includeディレクティブ
#include "money.h"

using namespace std;                         usingディレクティブ

// money.cxx : Moneyクラスのインプリメント
void Money::setYenRate(int Rate)
{
   YenRate = Rate;                           クラスの実装
}

void Money::setDollar(double doll)
```

```
{
   balance = (int)(YenRate * doll);
}

// Moneyクラスを使うプログラム本体
int main(int argc, char* argv[])
{
   int Rate;
   double value;
   Money* dv = new Money();
   cout << "1$ は円でいくら？>";
   cin >> Rate;
   dv->setYenRate(Rate);
   cout << "ドル（$）の価格は>";
   cin >> value;
   dv->setDollar(value);
   cout << "日本円の価格= " << dv->getYen() << endl;
   return 0;
}
```

main()

▼ リスト1.5 Money.h（クラス宣言やインターフェイス定義を行うヘッダファイル）

```
//
// money.h : Moneyクラスの宣言とインターフェイス
//
#ifndef money_H
#define money_H

// Moneyクラスの宣言
class Money {
public:
   Money() { balance = 0; }; // コンストラクタ
   ~Money() {}; // デストラクタ
   void setYenRate(int Rate);
   void setDollar(double doll);
   int getYen() { return balance; };
private:
   int balance; // 現在の残高(円単位)
   int YenRate; // 1$に対する円
};

#endif // money_H
```

同じヘッダにファイルを2度以上インクルードしないようにするための定数宣言と、すでにインクルードしたかどうかのチェックを行う。（＊も同じ）

クラスの宣言

＊

リスト1.5の#ifndef money_Hと#define money_Hおよび#endifは、同じヘッダファイルを2度以上インクルードしないようにするための仕掛けです。このファイルが一度でもインクルードされてプリコンパイルされるとmoney_Hが定義されます。再びこのファイルがインクルードされたときには、すでにmoney_Hが定義されているので#ifndef money_Hから#endifまでは無視されます。

03 C言語からC++の関数/変数を使う方法

C言語から呼び出すC++の関数や、C言語からアクセスするC++の変数には、extern "C"を付けて宣言します。

```
extern "C" type funcincpp ();
extern "C" var;
```

{ }を使うと、一連の関数や変数をまとめてextern "C"宣言できます。また、extern "C"が必要なのはC++のソースとしてコンパイルするときであり、同じソースをC言語のソースとしてコンパイルするときにはextern "C"は不要です。#ifdefを使うと、C++のソースとして扱うときだけextern "C"を有効にすることができます。

```
#ifdef __cplusplus
extern "C" {
#endif
   type var1;
   type funcincpp1 ();
   type funcincpp2 ();
   type funcincpp3 ();
#ifdef __cplusplus
}
#endif
```

C++の関数printmsg()を呼び出すC言語のソースプログラムの例をリスト1.6に示します。関数printmsg()のソースコードはリスト1.7に示します。extern "C"はC言語の呼び出し規約にしたいシンボルだけに付ける必要があるので、宣言を記述するヘッダファイルでは、リスト1.8に示すように#ifdefを使ってextern "C"を付けて宣言し、C++としてコンパイルするときだけextern "C"が有効になるようにします。

▼ リスト1.6 csrc.c

```
/*
 * csrc.c - C++の関数を呼び出すC言語ソースの例
 */
#include <stdio.h>
#include "cppsrc.h"

int main (int argc, char *argv[])
{
   puts("hello, C world");

   // C++の関数
   printmsg();

   return 0;
}
```

▼ リスト1.7 cppsrc.cpp

```
//
// cppsrc.cpp - C++の関数を含むモジュール
//
#include <iostream>
#define CPP
#include "cppsrc.h"

using namespace std;

void printmsg ()
{
   cout << "hello, C++ world¥n";
}
```

▼ リスト1.8 cppsrc.h

```
/*
 * cppsrc.h
 */
#ifndef CPPSRC_H
#define CPPSRC_H

#ifdef CPP
// C++のソースではextern "C"宣言する
extern "C" void printmsg ();
#else
// C言語のソースではextern "C"宣言は不要
void printmsg ();
#endif

#endif
```

この例は、C言語とC++の言語のソースを併用する方法を示す単純な例です。C++の標準出力ストリームcoutへの出力とC言語の標準出力stdoutを混在させることは一般には推奨されません。

01 C言語プログラムの構成要素

C言語やC++のプログラムは、主に、コメント、言語のキーワードと、ライブラリ関数やユーザー定義の関数、宣言した変数などを使って記述します。

C言語のソースプログラムを構成する主な要素は以下のとおりです。

コメント

C言語では/*から*/まではコメントとみなされます。また、現在のほとんどのコンパイラは、//以降行末までをコメントとして認識します。コメントの詳しい説明と例は、「02-01 基本要素」の「コメント」を参照してください。

言語キーワード

#includeや#defineのようなディレクティブ、ifやswitchのようなステートメント、intやdoubleのような型宣言に使う単語は、C言語のキーワードです。C言語のキーワードは再定義できません。また、キーワードと同じ名前を変数名などほかの目的に使うことはできません。

C言語のキーワードは以下のとおりです。

```
auto        double    int        struct
break       else      long       switch
case        enum      register   typedef
char        extern    return     union
const       float     short      unsigned
continue    for       signed     void
default     goto      sizeof     volatile
do          if        static     while
```

各キーワードの説明と例は、「02-02 ディレクティブと宣言」、「02-03 ステートメント」などを参照してください。

なお、ほとんどのコンパイラや開発ツールは、それぞれ独自にキーワードを追加しています。また、C/C++コンパイラの場合、C言語のソースプログラムの中でC++のキーワードを使えない場合があります。

識別子

識別子とは、プログラムで使う変数、型、関数などに付ける名前で、シンボルともいいます。

識別子の説明と例は、「02-01 基本要素」の「識別子」を参照してください。

ライブリ関数

ライブラリは関数を集めてまとめたものです。ライブリ関数は、特定のライブラリに含まれる関数を指します。

ライブラリは次の種類に分類することができます。

・言語の標準ライブラリ

printf()やgetc()のような、C言語の重要な関数が含まれる最も基本的なライブラリです。C++のプログラムはC言語の標準ライブラリ関数も利用できます。開発ツールによっては、ランタイムライブラリと呼ぶことがあります。

・ウィンドウシステムのライブラリ

X Window System（以下、X）やWindowsなどのウィンドウシステムが提供するプログラムを作るときに使います。たとえば、WindowsのGUIプログラムでウィンドウにメッセージを送るときには、Win32APIライブラリの関数SendMessage()を使います。

本書ではウィンドウシステムのライブラリについては概要だけを取り上げます。各ウィンドウシステムの詳しい内容は、各ウィンドウシステムの開発者用ドキュメントを参照してください。

・そのほかのライブラリ

OpenGLなどのサブシステムを使うときには、それぞれのライブラリを使います。また、開発ツールのユーザーである、アプリケーションやコンポーネントなどのプログラマも、自分で作った関数や既存の関数をまとめてライブラリとして作成し、利用することができます。

本書ではさまざまなサブシステムのライブラリの詳細については取り上げません。それぞれのサブシステムの開発者用ドキュメントを参照してください。

そのほかの関数

開発者は、自分で作った関数や既存の関数を使うことができます。また、作成した関数をまとめて独自のライブラリを作成することも可能です。

C言語プログラムの主な要素をリスト1.9に示します。

▼ リスト1.9　C言語プログラムの要素

```
/*
 * normalw.c - 標準体重を計算するプログラム
 */
#include <stdio.h>
#include <locale.h>

#define BMI 22
double CalcSW(int height);

int main(int argc, char *argv[])
{
   int height;
   setlocale( LC_ALL, "Japanese" );

   //ユーザーの入力
   printf("身長(cm)を入力してください>");
   scanf("%d" , &height);

   //標準体重を表示する
   printf("標準体重%.2f kg¥n" , CalcSW(height));

   return 0;
}

//標準体重を計算する
double CalcSW(int height)
{
  double h;
  h = height * 0.01;
  return (h * h* BMI);
}
```


ifやreturnはステートメントで、printf()やscanf()などは関数です。#defineや#includeのように#で始まるものはディレクティブです。プログラムを記述したりすでにあるプログラムを理解するときには、これらのカテゴリの違いを認識することが重要です。

02 C++プログラムの構成要素

C++プログラムでは、一部の例外を除いて、C言語の要素をそのまま使うことができます。加えてC++言語特有の次のような要素を使うことができます。

C++言語のキーワード

C++では、C言語のディレクティブ（#defineや#include）とステートメント（ifやreturnなど）に加えて、usingやnew、deleteなどのC++特有のキーワードを使うことができます。ディレクティブやステートメント、演算子については「02 C/C++のシンタックス」で説明します。

ライブリ関数

C++のプログラミングでは、原則的にC言語のライブラリ関数をそのまま使うことができます。たとえば、C++では文字列の出力にcout << s << endl;のようなコードを使いますが、C言語のputs(s);やprintf("%s¥n", s);なども使うことができます。ただし、C++でC言語の関数を使うときには注意を払うべきことがいくつかあります。そのような点については、「03 C言語リファレンス」の関数の説明の中で C++ マークを付けて説明します。

クラスライブラリ

C++ではC++のクラスライブラリを使うことができます。その中で特に重要なのは標準C++ライブラリです。標準C++ライブラリについては「04 C++リファレンス」で説明します。

C++プログラムの主な要素をリスト1.10に示します。

▼ リスト1.10　C++プログラムの要素

```
//
// ptrmap - main.cpp          ― コメント
//
#include <iostream>
#include <map>
#include <string>
                              ― C++のキーワード
using namespace std;

enum types {
    type_string,
    type_int
};
```

STLコンテナ

```
typedef map< string, void *, less< string > > Vars;
typedef map< string, int, less< string > > VarType;

void *str2void(string value)
{
   return ((void *)(new string(value)));
}

int main()
{
   Vars var;
   VarType vtype;

   // 変数名と値を保存する
   var["a"] = str2void("value of a");
   vtype["a"] = type_string;
   string value = "value of X";
   var["X"] = (void *)(new string(value));
   vtype["X"] = type_string;
   int n = 10;
   var["n"] = (void *)(new int(n));
   vtype["n"] = type_int;
   // 変数名と値を表示する
   Vars::const_iterator iter;
   VarType::const_iterator viter;

   for (iter = var.begin(), viter = vtype.begin();
      iter != var.end(); ++iter, ++viter )
   {
      void *pv = (*iter).second;
      int t = (*viter).second;
      if (t == type_string)
      {
         string s = *static_cast<string *>(pv);
         cout << (*iter).first << '¥t' << s << endl;
      }
      else
      {
         int k = *static_cast<int *>(pv);
         cout << (*iter).first << '¥t' << k << endl;
      }
   }
   return 0;
}
```

C++の演算子（new）

C++の演算子（::）

C++の演算子（<<）

C++のキーワード（static_cast）

03 C/C++とライブラリ

通常、C言語やC++のプログラミングでは、既存の関数ライブラリやクラスライブラリを利用します。また、自分で一連の関数を作成してライブラリにしたり、クラスライブラリを作って使うこともあります。多くの関数やクラスは特定のライブラリに含まれているので、プログラムを記述したりすでにあるプログラムを理解するときには、プログラムの中で使用する関数やクラスが属するライブラリの種類や名前を認識する必要があります。

基本的なC言語プログラミング

K＆R（参考書籍①）で説明されているような単純な入出力を伴う小さなプログラムを作成するときには、C言語の標準C言語ライブラリを使います。このライブラリは、通常、コンパイラに付属していて、多くの場合、デフォルトでプログラムにリンクされます。

一部のコンパイラでは、実数の演算を行うプログラムをビルドするときにはmathライブラリを明示的に指定するなど、特定のプログラムに対して特定のライブラリを明示的に指定する必要がある場合があります。

基本的なC++プログラミング

基本的で単純なC++プログラムを作成するときには、標準C++ライブラリを使います。また、C言語の標準ライブラリを使うこともできます。

C/C++プログラミングとGUIプログラミング

XやWindowsなどのGUIプログラムを作成する場合には、言語の標準ライブラリに加えて、XのGUIライブラリ（XlibやMotifなど）、あるいは、WindowsのWin32 APIと基礎クラスライブラリ（MFCなど）を利用します。

Win32API	ウィンドウ表示、プロセス管理、 メッセージ処理など
基礎クラスライブラリ	C++のクラスを活用したウィンドウ表示、 プロセス管理、メッセージ処理など
C/C++標準ライブラリ	基本的な操作など

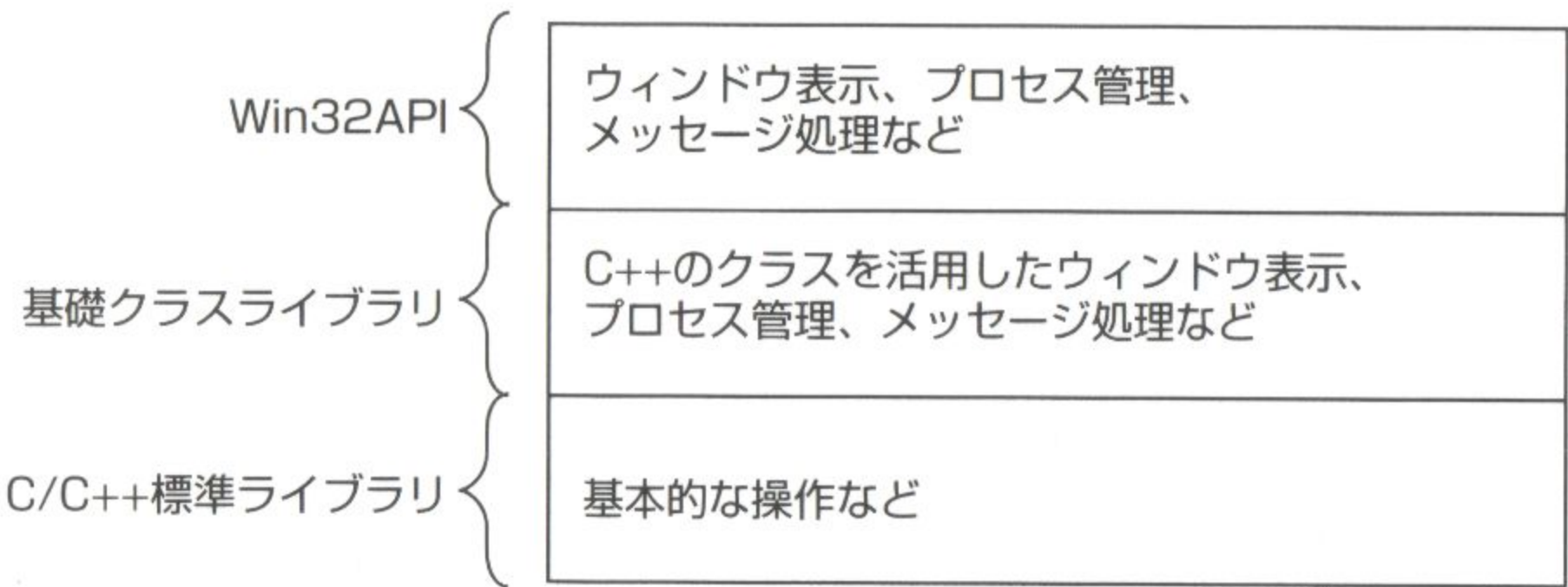
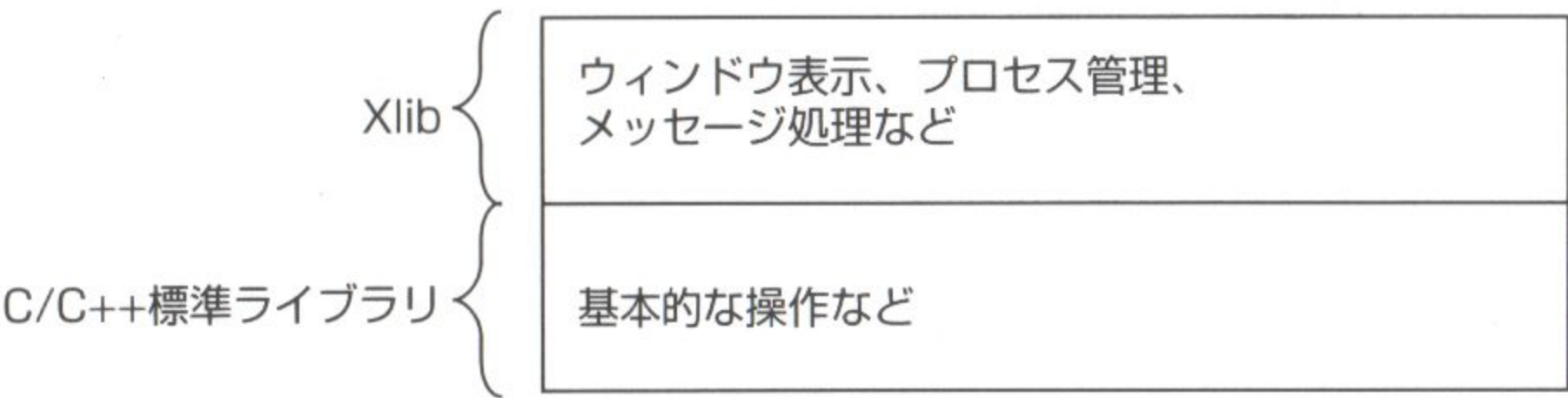



図1.4　WindowsのGUIプログラミングの概念（クラスライブラリを利用する場合）

Xlib	ウィンドウ表示、プロセス管理、メッセージ処理など
C/C++標準ライブラリ	基本的な操作など

● 図1.5 XのGUIプログラミングの概念（Xlibの場合）

プログラミング豆知識

C++におけるライブラリの選択

C++では、C言語のライブラリ関数とC++のライブラリ関数やC++のオブジェクトを使うことができます。しかし、同じカテゴリのライブラリに含まれるものを無秩序に混在させて使うことは望ましくありません。たとえば、入出力にC言語のstdio.hで定義されている入出力関数と、C++のcinやcoutを混在させると、入出力の順序などが予期しない結果になる可能性があります。

一般的には、C++のプログラムでは、第一にC++のライブラリに含まれるものを使い、さらに必要ならばC++でC言語関数を使うためのインクルードファイル（たとえば、cstdioやcctype、cmathなど）を使います。そして、stdio.hやctype.hなど、純粋なC言語の関数ライブラリのために定義されているものをC++のプログラムで直接使うのは最後の手段にするとよいでしょう。

01 基本的な開発手順

プログラマが作成したり編集するC言語やC++言語（以下、C/C++）のソースプログラムファイルは、テキストファイルです。プログラマは、最初にソースファイルをエディタで作成・編集します。そして、それをコンパイルして実行可能なプログラムファイルを作成し、実行します。ここではテキストエディタとC/C++コンパイラを使ってC/C++プログラムを開発するときの作業の流れとコンパイラの処理の過程を概説します。IDE（統合開発環境）を使って開発する場合でも、開発やデバッグを効率良く行うために、この基本的な操作の流れとコンパイルの各過程について知っておく必要があります。

ここでは、UNIX系のOSまたはDOS上でのコマンドラインコンパイラを使ったシンプルな開発手順を示します。

ソースプログラムを作成して実行するまでの手順は次のとおりです。

❶ テキストエディタを使ってC/C++のプログラムのソースファイルを作成したり編集する

編集したファイルは、C言語のプログラムの場合、通常、拡張子が.cであるファイル名を付けて保存します。C++のプログラムは、拡張子を.cxxか.cppにして保存します。

C/C++のプログラミングで使うファイルの拡張子については、付録A「C/C++プログラミングのファイル」を参照してください。

❷ ソースファイルをコンパイラでコンパイルする

通常使うコンパイラのコマンドは**gcc**、**g++**（C++）、**cc**（C言語）などです。
Visual C++のコマンドラインコンパイラはcl.exe、C++ BuilderやBorland C++コンパイラはBCC32.exeです。コンパイラごとのオプションや生成結果の違いについては、各コンパイラのドキュメントを参照してください。

単純なプログラムをコンパイルして実行する場合は、いずれを使っても操作方法や生成される結果はほぼ同じです。

プログラムを正しくコンパイルできない場合は、付録B「トラブル対策」を参照してください。

コンパイルが問題なく終了すると実行可能ファイルができます。

❸ 生成された実行可能ファイルを実行する

生成されたプログラムファイルを実行します。生成される実行可能なプログラムファイルの拡張子は、Windowsでは普通は.exeです。UNIX系OSでは（指定した場合）プログラム名と同じ名前か、（指定しない場合）a.outです。

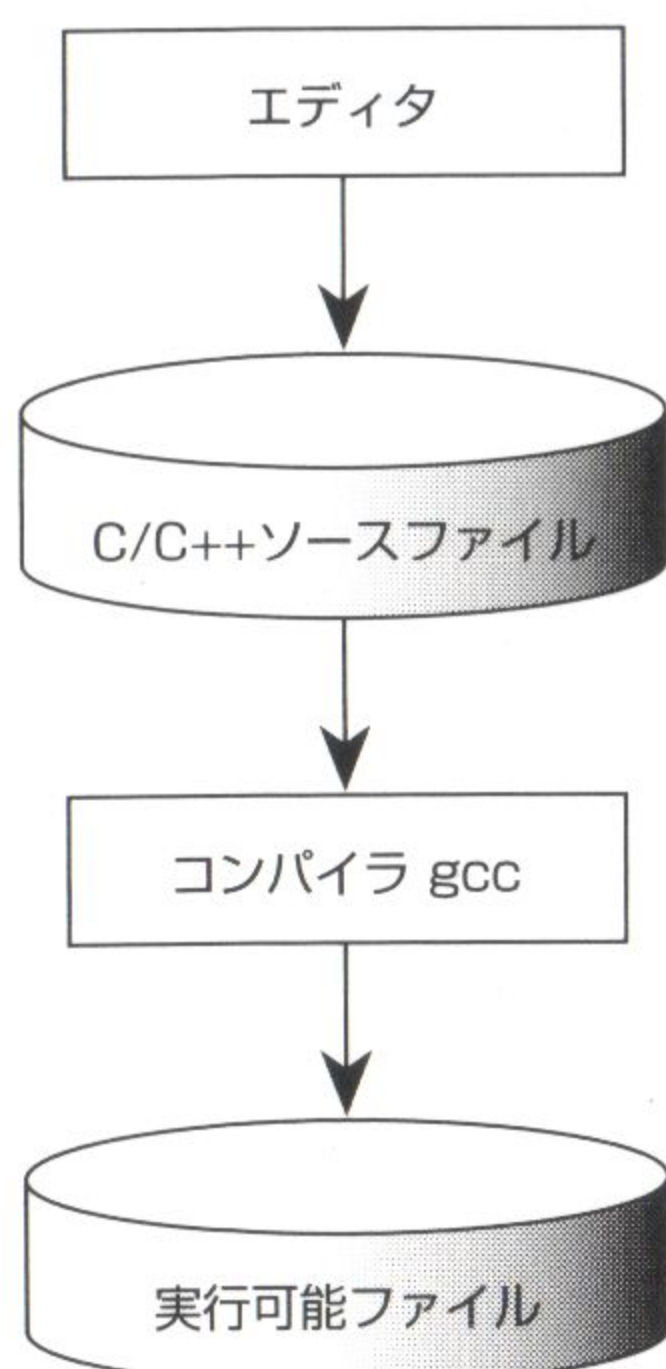



図1.6　C/C++プログラムのコンパイル手順の例

以上が基本的なコンパイルの手順です。このあと、プログラムをテストしてデバッグしますので、通常は手順❶～❸を繰り返す必要があります。一連の作業を効率良く行うためには、**make**と**makefile**を使います（参照→付録A、参考書籍②）。

コンパイルの過程

普通、単にコンパイル（またはビルド）と呼んでいる、ソースファイルから実行可能ファイルを生成する処理は、さらに細かく分けることができます。

実際に行われるコンパイルの過程は、図1.7に示すように、プリプロセス、コンパイル、リンクに分けることができます（図1.7は図1.6をさらに詳しく説明した図です）。

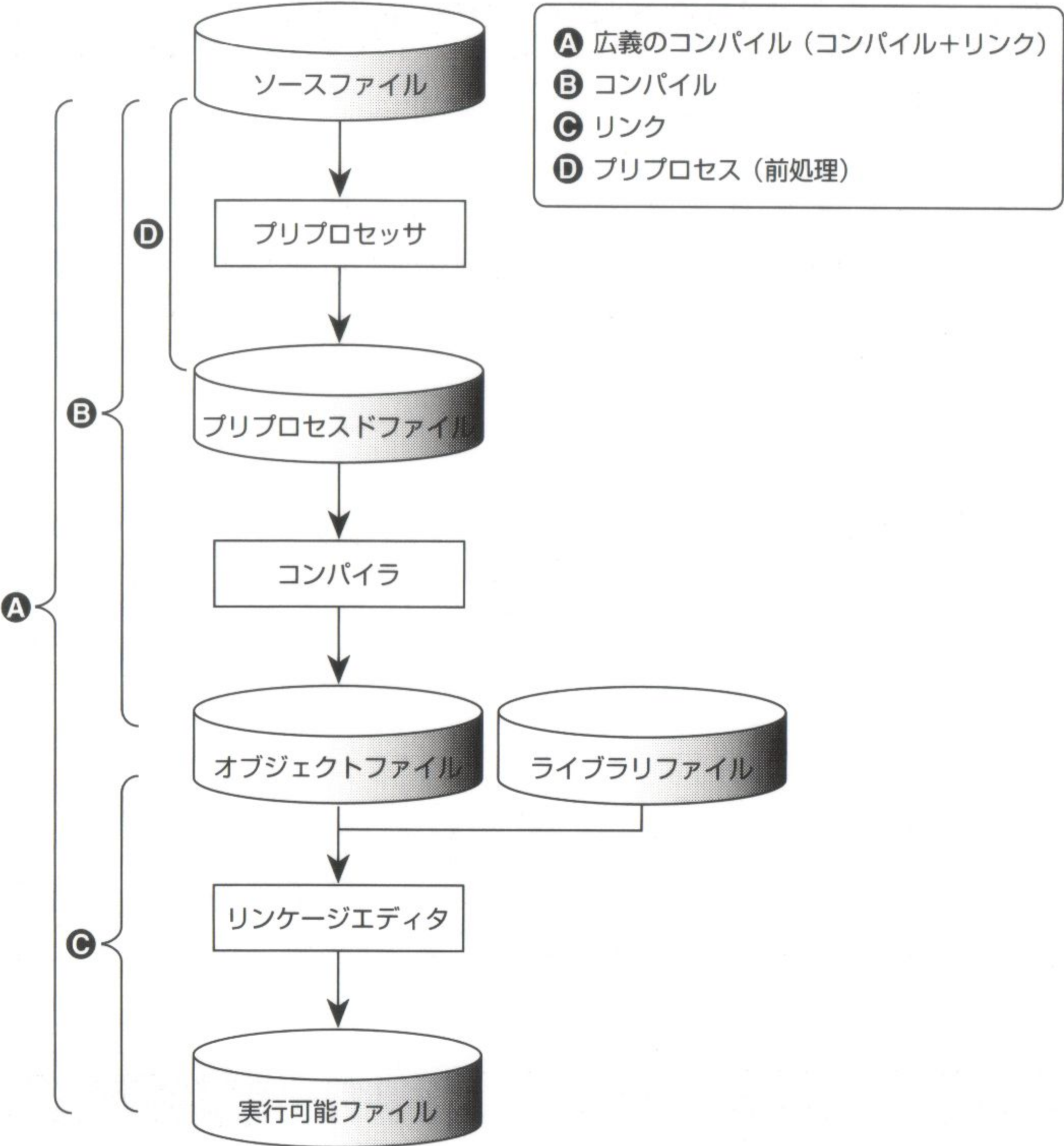
ソースファイルをコンパイルしてオブジェクトファイルを生成するプログラムをコンパイラといいます。ソースファイルを前処理（プリコンパイル）する部分だけを、特にプリプロセッサといいます。

オブジェクトファイルをほかのオブジェクトファイルやライブラリとリンクして実行可能ファイルを生成するプログラムを、リンカまたはリンケージエディタといいます。

前項で説明したコンパイルのコマンド**gcc**や**g++**は、正確にいえば、コンパイルとリンクを制御するコマンドです。プログラムを単にコンパイルして実行可能ファイルを作成するときには、コンパイルとリンクを制御するコマンドである**gcc**や**g++**だけを使ってコンパイルできますが、このようなコンパイルの詳しい過程を知っておくと、デバッグや効率的なコンパイラオプションの指定などで役立ちます。

01-04　プログラム開発の流れ

ソースファイル
プリプロセッサ
プリプロセスドファイル
コンパイラ
オブジェクトファイル
ライブラリファイル
リンケージエディタ
実行可能ファイル

Ⓐ 広義のコンパイル（コンパイル＋リンク）
Ⓑ コンパイル
Ⓒ リンク
Ⓓ プリプロセス（前処理）

● 図1.7　C/C++プログラムのコンパイル手順

プログラムの開発例

ここでは、UNIX系のOSまたはDOS上でコマンドラインコンパイラgccあるいはccを使ってC言語の小さなプログラムをコンパイルする例を示します。

ここで作成する小さなプログラムは、「hello, C world」と「Hit Enter Key」という文字列を表示して、ユーザーがキーボードのEnterキーを押すまで待って終了します。このプログラムを正しくコンパイルして実行できれば、C言語のプログラミングの基本的な環境が整っていることを確認できます。

ディレクトリの作成

通常、プログラムを作るときには、最初にそのプログラム専用のディレクトリを作成します。この場合、helloという名前のプログラムなので、**mkdir**コマンドでサブディレクトリhelloを作成します。そして、そのあとで、**cd**コマンドでカレントディレクトリをhelloにします。

▼ UNIX系OSの場合

```
# mkdir hello
# cd hello
```

▼ WindowsやDOSの場合

```
> mkdir hello
> cd hello
```

プログラムの編集

エディタを起動してリスト1.11のプログラムを入力します。

▼ リスト1.11　hello.c

```
/*
 * hello.c
 */
#include <stdio.h>

int main (int argc, char *argv[])
{
```

```
    puts("hello, C world");

    printf("Hit Enter Key");
    return getchar();
}
```

プログラムを入力したら、hello.cという名前を付けてファイルに保存します。

コンパイル

コンパイラgccまたはcc、cl、BCC32などのコンパイラのコマンドを使ってコンパイルします。

一般的に基本的なコンパイル方法は、次のようにコンパイラの名前のあとにオプション-oと実行可能ファイル名を指定し、さらにソースフィアル名を指定する方法です。

```
gcc -o hello hello.c
```

このコマンドラインの-oは多くのC/C++コンパイラの出力ファイル名を指定するためのオプションです。

この例の場合、コンパイルが問題なく終了すると、指定したとおりhelloという名前の実行可能ファイルが生成されます。

実行

エラーメッセージが表示されなければ、コンパイルが成功しているので、「./hello」と入力し、Enterキーを押して実行します。

▼ UNIX系OSの場合

```
# ./hello
hello, C world
Hit Enter Key
```

./はカレントディレクトリのファイルであることを示します。環境変数PATHにカレントディレクトリ（.）が含まれていれば、「hello」と入力してEnterキーを押すと実行することができます。

▼ WindowsやDOSの場合

```
> hello
hello, C world
Hit Enter Key
```

プログラムを正しくコンパイルしたり実行できない場合は、付録B「トラブル対策」を参照してください。また、Visual C++やC++ BuilderなどのIDEを使う場合は、それぞれの製品のドキュメントを見てhelloプログラムを作成してみてください。

01 コンピュータと日本語

現在ではC言語やC++のプログラムの中で日本語をあまり意識せずに使うことができます。しかし、ときには日本語の文字を使ったために、いわゆる文字化けやそのほかのトラブルが発生することがあります。ここでは日本語を使うC/C++プログラミングに特有なことについて簡潔に取り上げます。

コンピュータの文字

現在使われているコンピュータは、もともとは英語圏で開発が進められました。そのため、当初は英数字と一部の記号だけを扱っていて、日本語を扱うことは想定されていませんでした。

基本的には、コンピュータではASCIIコードという1バイトで表現できる文字コードが使われてきました。C言語とC++のプログラミングでも、当初は、文字は1バイトで表現でき、ASCIIコードを使うということが前提でした。

その後、コンピュータで日本語を扱う必要性から、日本語を表現するための文字コードが考案されました。日本語は文字数が多いので、1バイトでは表現できません。そのため、2バイト以上のバイトで1文字を表します。C言語とC++のプログラミングでも、日本語の文字は2バイト以上の連続するバイトで表現します。

日本語の表現

日本語の文字は、Unicode、JIS、シフトJIS、日本語EUCなどで表現されます。Unicodeにはエンコーディング（符号化）の方法が複数あり、UTF-8やUTF-16がよく使われます。

Unicodeやシフト JISなどでは、原則的には1文字を2バイトの情報で表現します。C言語とC++のcharのサイズは普通は1バイトですから、日本語の文字1文字はcharを2個使って表現します。

ネットワークで国境を越えてコンピュータが接続され、国や言語を越えて情報を交換しなければならなくなったので、最近は、コンピュータのあらゆる分野で文字コードとしてUnicodeが使われるようになってきています。そのため、C/C++ではプログラム内部ではワイド文字（2バイトで1文字を表すUnicode文字）を使うことが多く、そのための型wchar_tが定義されています。一方、プログラムの外部（ファイル、ネットワークを使った転送など）では、日本語を扱うときには伝統的にシフトJISや日本語EUCなどが使われてきましたが、最近はUTF-8が多く使われるようになっています。

なお、UTF-8エンコーディングのUnicode文字は1文字を1〜4バイト（最長6バイトですが6バイトを使う文字はまだ定義されていません）の可変長の数値で表します（Unicodeだからといって常に2バイトで表現されるわけではありません）。

ロケール

地域や国によって変わるのは文字だけでなく、日時や金額の表記なども地域や国によって変わることがあります。日時や金額の表記のような地域や国によって変わることは、ロケール（locale）で表現します。

日本語を使うC言語とC++のプログラムでは、次のようにしてロケールを日本に設定します。

```
setlocale( LC_ALL, "Japanese" );
        // または
setlocale(LC_ALL, "ja_JP.ujis");
        // または
setlocale(LC_ALL, "ja_JP.EUC");
```

LC_ALLはロケールに関するあらゆる設定を変更することを意味します。このほかに、たとえば、LC_CTYPE（文字列処理関数のロケール）、LC_NUMERIC（数値の表現）、LC_MONETARY（金額の表現）、LC_TIME（日時の表現）など、個別にロケールを指定することもできます。

なお、英数字を処理する関数には、ロケールを日本に設定すると正常に機能しない場合があるものがあります。たとえば、システムによっては、localeをC以外にすると、関数isalnum()が0x80以上の文字に対してtrueを返すことがあります。そのような場合にはsetlocale (LC_ALL, "C");またはsetlocale(LC_CTYPE, "C");を使います。

ワイド文字対応関数

ISO/ANSI Cでは、プログラム内部でワイド文字にwchar_t型のUnicodeを使います。そのため、ワイド文字対応関数が用意されています。たとえば、ASCII文字のためのprintf()に対してワイド文字用にwprintf()が、fputc()に対してfputwc()が用意されています。

次に、wchar_tおよびsetlocale()とワイド文字対応関数を使ったプログラムの例を示します。

```
/*
 * widehello.c
 */
#include <stdio.h>
#include <locale.h>

int main (int argc, char *argv[])
{
   wchar_t s[] = L"C言語でこんにちワン";

   setlocale(LC_ALL, "japanese");

   wprintf(L"%s¥n", s);

   return 0;
}
```

なお、C言語には、マルチバイト文字やマルチバイト文字列に対応した関数も定義されています。マルチバイト対応関数は、関数名の最初か途中にmbという文字列が含まれます。

日本語を含むプログラムのコンパイル

一般的には、そのシステムのネイティブな文字コードを使って作ったソースコードは、そのシステムのコンパイラでそのままコンパイルして、そのまま実行できるのが普通です。たとえば、日本語EUCの環境では、日本語EUCでプログラムのソースコードを作成し、普通にコンパイルして、日本語EUCの環境で実行すれば問題は発生しません。

日本語を含むプログラムをコンパイルして実行したときに何らかのトラブルが発生したら、そのプログラムを実行する環境のネイティブな文字コード（エンコード）を調べて、ソースファイルをその文字コードに変換して再コンパイルして実行してみてください。

Windowsではシフト JISが、UNIX系OSではUTF-8が使われることが多くなってきていますが、依然としてさまざまな文字コードが使われています。ファイルの文字コードを変換するときには、**iconv**、**recode**または**nkf**などのコマンドを使うことができます。

02：C/C++の
シンタックス

この章では、プログラミング言語としてのC++とC言語の基礎を説明します。C言語とC++の言語構造や規則、例外処理など、どのような種類のプログラムのプログラミングでも必要な基礎的なことを解説します。

基本要素

ここではC言語プログラムの最も基本的な要素である、空白（ホワイトスペース）、コメント、識別子、定数、変数などについて説明します。ここで解説することは、ほとんどどのような種類のプログラムを記述する際でも必須の事項です。

空白

空白とみなされる、スペース、タブ、改行、復帰などの総称をホワイトスペースと呼びます（改行、復帰はホワイトスペースに含まない場合もあります）。ここではホワイトスペースを「空白」と表記します。

C/C++では、以下のような場合を除いて、原則的に空白をいくつでも自由に挿入したり削除できます。

・キーワードや変数などの名前の中には空白は挿入できない

たとえば、MyNameという名前を変数や関数などの名前として使うことができますが、空白を含むMy Nameのような名前は定義できません。

・トークンの前後にある必要不可欠な空白は削除できない

トークン（言語要素）の区切りとして必要な空白は削除できません。たとえば、ifやreturnなどのキーワードの前後には少なくとも1つ以上の空白が必要です。

・2文字で意味をなす演算子は、文字間に空白を入れることができない

たとえば、等価であるかどうか示す演算子==は2文字で1つの意味をなす演算子なので、= =のように2つの文字の間に空白を入れることはできません。同様に空白を間に入れることができない2文字以上の演算子には、たとえば、不等であることを示す演算子!=、インクリメント演算子++、デクリメント演算子--などがあります。

・文字列リテラルの中の空白は、挿入した数だけそのまま評価される

たとえば、空白を文字と文字の間に5個入れれば、空白5個ぶんだけスペースがあきます。一方、ソースコードのインデントに使う空白は、3個から5個に変更しても、ソースコードの外観が変わるだけで、プログラムの動作や効果には影響を与えません。

インデント

ソースコードを読みやすくするために、行の先頭をほかの行より右側に表示する目的で、行の始めに空白を入れることをインデント（字下げ）といいます。

たとえば、forやifのブロックは、次のようにするとどこまでがブロックの範囲（{ }で囲まれた範囲）であるかわかりにくく、論理的にも追跡しにくいコードになります。

```
for (i=0; ;i++) {
fprintf(stdout, ">");
fgets(buff, 50, stdin);
buff[strlen(buff)-1] = '¥0';
if (strlen(buff)<1) {
n = i;
break;
}
strcpy(org[i], buff);
}
```

これは、次のようにインデントすると、とても明確になります。

```
for (i=0; ;i++)
{
   fprintf(stdout, ">");
   fgets(buff, 50, stdin);
   buff[strlen(buff)-1] = '¥0';
   if (strlen(buff)<1)
   {
      n = i;
      break;
   }
   strcpy(org[i], buff);
}
```

インデントの幅は自由ですが、通常は、1つの字下げの単位を、スペース2個、3個、4個、8個ぶんのいずれかにするのが普通です。以前はインデント1つの幅としてスペース8個または4個ぶんにすることが一般的でしたが、現在はスペース3個ぶんにすることもよくあります（本書では紙面を節約することを主な目的として、インデントをスペース3個ぶんにします）。

コメント

コメントはソースプログラムの中に記述できる注釈です。コメントはプログラムの実行に影響を与えません。

コメントにはC言語スタイルのコメント（/* ... */）とC++スタイルのコメント（// ...）があります。

C言語スタイルのコメント

C言語では/*から*/まではコメントとみなされます。たとえば、次のように記述します。

```
/* これはC言語スタイルのコメント*/
```

C言語スタイルのコメントの/*と*/の間には、改行を含めることができます。ですから、次のように2行以上に渡るコメントを記述することが可能です。

```
/* C言語スタイルのコメントなら、
   2行以上のコメントも記述できる*/

/*
 * こんなふうにコメントブロックを
 * 記述することもできる。
 */

/*********************************
 * これでもOK
 *********************************/
```

コメントの中の*や****は外見を整えるためのものです。

C言語スタイルのコメントをネストすることは多くの処理系で認められていません。C言語スタイルのコメントのネストとは、「/*これは/*コメント*/のネスト*/」のように、/*…*/の中に/*…*/のコメントを記述することです。

C++スタイルのコメント

C++のコメントは、//以降、行末までです。

```
// これはコメント

if (a) { // このように行の途中から記述することもできる
```

現在のほとんどのコンパイラは、ソースファイルの種類にかかわらず、C言語スタイルのコメントとC++スタイルのコメントの両方をサポートします。そのため、多くの場合、C言語のソースプログラムでもC++スタイルのコメントを使用可能です。

ただし、ごく一部のC言語コンパイラはC++スタイルのコメントをサポートしません。また、コンパイル時のオプションの指定によって、有効なコメントの形式が変わる処理系もあります。どのC言語コンパイラでもコンパイルできるC言語プログラムとして記述したい場合は、C言語スタイルのコメントを使うと良いでしょう。C++ではC++スタイルのコメントに加えてC言語スタイルのコメントも常に使用可能です。

識別子

識別子とは、プログラムで使う変数、型、関数などに付ける名前で、シンボルともいいます。

ここで説明する識別子に関する説明は、変数名、関数名、新しく定義したクラス名、構造体名など、プログラミングで使うほとんどの名前に原則的に当てはまります。また、gotoステートメントのジャンプ先として使う「ラベル」と呼ぶ特殊な識別子も使うことができます。

名前を付けるときには、容易に理解できる範囲で短く簡潔な名前を付けるべきです。識別子の名前付けの規則は処理系やオプションの指定によって異なることがありますが、識別子を付けるときには、基本的には次のような点に注意を払う必要があります。

- 名前には、アルファベットの大文字（A～Z）と小文字（a～z）、数字（0～9）、アンダースコア（_）を使うことができます。

ただし、最初の文字は、A～Z、a～z、あるいは下線記号（_）でなければなりません。最初の文字が数字であってはなりません。wanwan_dogは有効な名前ですが、11_dogは無効です。
なお、C++では文字$を名前に使うことができるなど、コンパイラによってはこのほかの文字も使うことができる場合がありますが、特に理由がない限り、ソースコードの互換性の点からほかの文字は使わないほうがよいでしょう。

・識別子には、大文字小文字も含めてキーワードと同じ名前を付けることはできません。たとえば、ifは識別子として使うことはできません。

・識別子の長さは長過ぎないようにする必要があります。

有効な識別子の長さは規格や処理系によって異なります。たとえば、ANSI Cでは外部識別子として6文字まで、内部識別子（関数内の識別子）として31文字までの名前が有効です。一般的なC++コンパイラでは、外部識別子は32文字、内部識別子は250文字前後までの名前が有効です。これらの制限を超える長さの名前は、制限を越える部分が無視される場合があります。たとえば、6文字までが有効である場合、wanwandogとwanwancatとが6文字に切り詰められます。その結果、処理系内部や生成されるファイルの中で、どちらも同じwanwanとして扱われるか、wanw_1とwanw_2のように自動的に名前が変更されて処理されることがあります。
また、C++のプログラムは同じ名前の関数を定義できることやシンボリックデバッグ情報を付けるためなどの理由で、コンパイラがコンパイル時に名前を変更して長くすることがあります。その際にも名前の長さの制限が適用されて、制限を越える部分が切り捨てられることがあります。
以上のような理由から、関数や変数などの名前があまり長過ぎることのないように配慮するべきです。

定数

定数は、プログラムの実行中に変わらない値です。文字、文字列、数値を定数として指定できます。
定数の定義方法は、#defineを使う方法と、キーワードconstを使う方法があります。また、一連の整数定数を定義したいときにはenumを使って列挙型として定義します。

#define

#defineは定数を定義するプリプロセッサディレクティブです。たとえば、次のように使います。

```
// 文字定数（文字定数の定義には''を使う）
#define DEFCHAR 'A'

// 文字列定数（文字列定数の定義には""を使う）
#define MYNAME "Pochi Dog"

// 数値の定数
#define PI 3.14
```

#defineを使って定義する定数は、慣例としてそれが定数であることが明確になるように、MYNAMEのようにすべて大文字の名前にすることがあります。しかし、これは慣例であり、定数名を常にすべて大文字にしなければならないわけではありません。

強固なプログラムを開発するためには、可能な限り、#defineの代わりに、次に説明するキーワードconstを使って定数を定義してください。constで定義した値は型がチェックされるので、より安全確実なプログラミングが可能です。#defineはDEBUGのようなプリプロセッサが使う特別な定数を定義するときに主に使います。

const

キーワードconstは変数の値が不変であることを示し、文字、文字列、数値のほか、ポインタ宣言にも使うことができます。

constを使って宣言した値は、プログラムの実行中に変更することはできません。constで宣言したデータのポインタを関数の引数として使っても、その引数を受け取った関数は値を変更できません。

```
// 文字定数（文字定数の定義には''を使う）
const char c = 'A';

// 文字列定数（文字列定数の定義には""を使う）
// stringは標準C++のクラス
const string MyName = "Pochi Dog";

// 実数値の定数
const int pi = 3.14;

// 整数値の定数
const int n = 5;
char name[256] = "Pochi";

// ポインタ宣言。定数変数のポインタしか代入できない
char *const pname = name; // 定数ポインタ
```

C言語のコンパイラの中には、constをサポートしないものがあります。その場合は#defineを使います。

文字列定数

文字列定数は文字列リテラルを使って定義します。そのため、文字列リテラルを文字列定数と呼ぶこともあります。定義する文字列リテラルは" "で囲みます。

```
const char dogname[] = "Hyuga Pochi";
```

空文字列は、" "で定義します。

```
char s[256] = ""; // sは空文字列
```

エスケープシーケンス

通常、文字として出力できない文字や特別な意味を持つ文字をソースコード上で表現するときには、円記号（¥）またはバックスラッシュ（\）と特定の文字か数字の組み合わせを使います。この文字を連結したもの（シーケンス）をエスケープシーケンスといいます。エスケープシーケンスは、ソースコード上では2文字ですが、1文字とみなします。

文字列リテラルの中でダブルクォーテーション（"）を使いたいときや、文字定数の中でシングルクォーテーション（'）を使いたいときにもエスケープシーケンスを使います。

```
// ¥nは改行のエスケープシーケンスなので、AとBそれぞれのあとで改行される
printf("A¥nB¥n");

// 文字列の中に"を含めるときには¥"にする
puts("文字列は¥"で囲みます");

putchar('¥''); // 'が表示される
```

ANSIエスケープシーケンスを表2.1に示します。

▼表2.1 ANSIエスケープシーケンス

エスケープシーケンス	意味
¥a	ビープ音（アラート）
¥b	バックスペース
¥f	フォームフィード
¥n	改行
¥r	キャリッジリターン（復帰）
¥t	水平タブ
¥v	垂直タブ
¥'	シングルクォーテーション（引用符）
¥"	ダブルクォーテーション（二重引用符）
¥¥	円記号（環境によってはバックスラッシュ\が表示される）
¥?	クェスチョンマーク（文字）
¥ooo	8進表記のASCII文字
¥xhhh	16進表記のASCII文字

円記号（¥）は、環境によってはバックスラッシュ（\）として表示されます。どちらも文字コードは同じで、外観だけが異なります。本書では円記号を使います。なお、円記号（¥）またはバックスラッシュ（\）は、行を連結する文字としても使われます。

```
/*
 * ensymbol.c
 */
#include <stdio.h>

int main (int argc, char *argv[])
{
   char s[] = "長い長い文字列リテラルは¥
¥¥記号を使って2行に分けることができます。";

   printf("%s¥n", s);

   return 0;
}
```

02-01　基本要素

変数

変数は、値を保存するためのシンボルです。変数を宣言するには、型と名前、および必要に応じて配列のサイズを指定します。

```
// 整数変数の宣言
int i;

// 文字変数の宣言
char c; // 文字1文字を保存する

// 文字列変数の宣言
// 文字配列として宣言する
char name[256];

// ポインタ変数の宣言とメモリの確保
char *ps;
ps = (char *)malloc(256);
```

変数名を付けるときには次のような慣例に従うことがよくあります。

・単純な役割の整数変数には、i、j、kなどを使う

ループのカウンタのような単純な役割の変数には、i、j、kなどを使うことがよくあります。

```
for (int i =0; i++; i<n)
```

・役割を端的に表す名前を付ける

たとえば、文字列バッファから取り出した1文字を一時的に保存するための単純な役割の文字変数にはcを、文字列変数にはsやstrを、バッファにはbufやbuffなどを使うことがよくあります。

・言葉をつなげて名前を作るときには、アンダースコアを入れるか、大文字小文字を適切に使う

```
char MyFavoritSong[256];

int n_box;
```

値と型

言語の基本的なデータ型のサイズと数値の範囲などは、処理系に依存します。C/C++の基本的な型は、整数、浮動小数点数（実数）、voidの3種類に分けることができます。また、整数型を文字型と整数型に分けて考えることもあります。

整数

整数に属する型は、整数と文字を保存するための型です。整数の型を表2.2に示します。

▼表2.2 整数の基本型

型	内容	サイズ※
char	文字を1文字保存する。また、0～255までの数値を保存することもある。 C言語で文字列を保存したいときにはcharの配列を使うのが一般的。	8
signed char	符号付き文字を保存する。	8
unsigned char	符号なし文字を保存する。	8
short int	整数を保存する。通常は単にshortと記述しても同じ。 また、intと同じである処理系もある。	16
signed short	符号付き整数を保存する。	16
unsigned short	符号なし整数を保存する。	16
int	整数を保存する。	32
signed int	符号付き整数を保存する。通常、signed intはintと同じ。	32
unsigned int	符号なし整数を保存する。	32
long	大きな整数値を保存する。	64
signed long	大きな整数値を符号付きの値として保存する。 通常、signed longはlongと同じ。	64
unsigned long	大きな整数値を符号なしの値として保存する。	64

※サイズは32ビットのシステムの標準的な値で、単位はビット。
実際のデータサイズは主にCPUのアーキテクチャに依存し、コンパイラによっても異なります。
データサイズはsizeof演算子を使って調べることができます。

コンパイラの種類やオプションの指定によって、signedまたはunsignedが付けられていなくても、特定のデータ型をデフォルトでsignedまたはunsignedとして解釈する場合があります。たとえば、charはコンパイラの種類やオプションの指定状況によって、signed charまたはunsigned char型として扱われます。

浮動小数点数

浮動小数点数型には、1.23や0.456のような実数値を保存します。

C/C++の浮動小数点数の型を表2.3に示します。

▼ 表2.3 浮動小数点数の基本型

型	内容	典型的なサイズ
float	最小の浮動小数点型。	4バイト
double	サイズがfloat型以上で、long double型以下の浮動小数点型。 C/C++ではdoubleが実数演算の際のデフォルトの型。	8バイト
long double	サイズがdoubleと等しい浮動小数点型。型としては異なる。コンパイラ（WindowsではVisual C++またはMicrosoft C/C++のバージョン）の種類によって内部表現が異なる場合がある。	8バイト

C/C++では、実数演算はdoubleで行われます。これは精度を維持するためです。

たとえば、次のようなfloat型の値を使う式であっても、演算はdoubleで行われ、結果がfloatに変換されて代入されます。

```
float n, v;
scanf("%f", &n);
v = 1.2 * n;
printf("v=%8.3f¥n", v);
```

C/C++では実数は原則的にdoubleで扱うと考えることができますが、いくつか例外があります。たとえば、scanf()を使って実数を入力するときには、この関数は常にfloatの値を受け取ります。

void型

voidは主に関数やメソッドの宣言で、値がないことを示すときに使います。

たとえば、次の関数宣言は、関数func()には引数がなく、返す値もないことを示します。

```
void func(void);
```

void型の変数は作成できませんが、void型のポインタ変数（void *）は作成できます。void型のポインタは任意の型のデータを指すことができます。

```
// 任意の型の値のポインタを保存する
void *pv;

// 文字列のポインタを保存する
pv = (void *)(new string(value));
```

処理系やライブラリ独自の型

C/C++では、typedefや#defineを使って型を宣言することができます。このことを利用して、各処理系やライブラリは言語仕様には含まれていない、さまざまな型を定義していることがあります。

型定義は、通常、ヘッダファイル（.h）の中で行われています。ですから、ライブラリのヘッダファイルを調べることで、独自に定義されている型を知ることができます。

たとえば、次のような固有の型が使用可能である場合があります。

▼表2.4　処理系やライブラリに独自の型の例

型	内容
intn	サイズ付きの整数。nはビット数で表した整数変数のサイズで、8、16、32、64など。たとえば、int64は64ビットの整数型を表す。
BOOL	ブール値の型。true/falseの値を保存する。
BSTR	32ビットの文字ポインタ。
BYTE	8ビットの符号なし整数。unsigned charと同じ。
COLORREF	カラー値として使われる32ビット値（Windows）。
DWORD	WORDの2倍のサイズの整数。
LONG	通常は32ビットの符号付き整数。
LPARAM	パラメータ（引数）として使われるLONG（32ビット）の値（Windows）。
LPCSTR	定数文字列のLONG（32ビット）のポインタ（Windows）。
LPSTR	文字列のLONG（32ビット）のポインタ（Windows）。
LPVOID	型が指定されていないLONG（32ビット）のポインタ（Windows）。
LRESULT	関数が返すLONG（32ビット）のポインタ（Windows）。
UINT	unsigned int
WORD	16ビットの符号なし整数。

※（Windows）はWindowsでよく使われる型を示しています。

ディレクティブと宣言

ここでは、プログラムの実行のコンパイルなどに使われるC言語とC++のディレクティブや、宣言のための以下のさまざまなキーワードをアルファベット順に説明します。

項目	機能	
#define	マクロの定義	
#else	条件コンパイルのためのディレクティブ	
#endif	条件コンパイルのためのディレクティブ	
#ifdef	条件コンパイルのためのディレクティブ	
#include	ソースプログラムのインクルード	
auto	ローカル変数であることを示す記憶クラス指定子	
const	定数の定義	C++
enum	列挙型の定義	
explicit	暗黙の型変換が行われないようにするための指定子	C++
extern	外部シンボルであることを示す記憶クラス指定子	
mutable	値を変更できるようにする記憶クラス指定子	C++
namespace	名前空間の宣言	C++
register	レジスタ変数であることを示す記憶クラス指定子	
static	静的なシンボルであることを示す記憶クラス指定子	
struct	構造体を宣言する	
typedef	名前の定義	
union	型の異なる値を保存する構造を作る	
unsigned	符号なしとして宣言する	
using	使用する名前空間の指定	C++
void	型または引数がないことを示す	
volatile	変更可能なオブジェクトであることを示す	

※ C++ は原則的にC++だけで使うことができる項目です。
C言語では使えません（一部の処理系ではC言語でも使えるものがあります）。

01 マクロの定義 #define

書式　#define *id str*
　　　#define *id* [(*opt*, ... , *opt*)] *str*

パラメータ　*id* ………識別子。
　　　str ………定義する文字列。式も可能です。
　　　opt ……マクロの引数。

●解説

定数やマクロを定義するときに使います。マクロは、普通、ソースコードがプリプロセスされるときに展開されるので、関数を呼び出すより高速ですが、複雑なマクロの場合は生成されるコードサイズが大きくなる傾向があります。1行では書ききれないような長いマクロを定義したいときには、行継続文字¥を使って行をつなげます。

#defineは、普通はプリプロセッサで単純な置き換え作業として処理されます。そのため、予期しない置き換え結果がバグの原因とならないように、マクロの定義文字列全体をかっこ( )で囲みます。このような予期しない置き換え結果を「マクロの副作用」と呼ぶことがあります。

●例

次の例は、#defineを使って整数の定数としてMAX_VALUEと、マクロMAX(a, b)を定義して使う例です。

```
#include <stdio.h>

// 定数の定義
#define MAX_VALUE 99

// マクロの定義
#define MAX(a, b) (a>b ? a : b)

int main(int argc, char* argv[])
{
   int v;
   scanf("%d", &v);

   printf("%dと%dで大きいほうの値=%d¥n", v, MAX_VALUE, MAX(MAX_VALUE, v));

   return 0;
}
```

● **備考**

ほとんどのコンパイラで、シンボルやマクロの定義にコンパイラのオプション（一般的には-D）を使うことができます。

02 条件コンパイルのためのディレクティブ #else

書式	#else statements

● **参照**→#ifdef

03 条件コンパイルのためのディレクティブ #endif

書式	#endif statements

● **参照**→#ifdef

04 条件コンパイルのためのディレクティブ #ifdef

書式	#ifdef *symbol* 　*stat1* #else 　*stat2* #endif
パラメータ	*symbol*…定義されているかどうか調べるシンボル。 *stat1*……*symbol*が定義されているときに実行するステートメント。 *stat2*……*symbol*が定義されていないときに実行するステートメント。

● **解説**

シンボルが定義されていたら#ifdefの次の行から#elseまたは#endifまでのステートメントがコンパイルされます。シンボルが定義されていなくて#elseがあれば#endifまでのステートメントがコンパイルされます。シンボルが定義されていなくて#elseがなければ、#ifdefの次の行から#endifまでのステートメントはコンパイルされないで無視されます。

シンボルの定義には#defineか、コンパイラのオプション（一般的には-D）で指定します。

#ifdef～#else～#endifは、普通はプリプロセッサで処理されます。

05 ソースプログラムのインクルード #include

書式	①#include <*fname*> ②#include "*fname*"
パラメータ	*fname* …インクルードする（取り込む）ファイル名。 必要に応じてディレクトリも指定できます。

●解説

#includeディレクティブは、ソースプログラムのこのディレクティブの位置に、引数で指定したファイルの内容を取り込みます。

#includeディレクティブは、指定したインクルードファイルの全内容で置き換えられます。つまり、#include文#include <abc.h>があると、#include <abc.h>の行の位置にファイルabc.hの内容が挿入されます。#include <abc.h>そのものは削除されたものと解釈されます。

#includeでインクルードするファイルの中で別のファイルをインクルードする（#includeをネストする）こともできます。たとえば、上の例では、ファイルabc.hの中で別のファイルをインクルードしていてもかまいません。

●ファイルの検索順序

< >で囲まれたヘッダファイルは、通常、次の順序で検索されます。

①インクルードディレクトリを指定するコンパイラオプション（普通は/I）で指定されたパス
②インクルードディレクトリを指定する環境変数（普通はINCLUDE）で指定されたパス

" "で囲まれたヘッダファイルは、通常、次の順序で検索されます。

①#includeディレクティブを含むファイルのディレクトリ
②#includeでインクルードするファイルのディレクトリ
③カレントディレクトリ
④インクルードディレクトリを指定するコンパイラオプション（普通は/I）で指定されたパス
⑤インクルードディレクトリを指定する環境変数（普通はINCLUDE）で指定されたパス

一般的には、コンパイラや基本的なライブラリが提供しているヘッダファイルのファイル名を囲む記号には< >を使い、そのプロジェクト特有のヘッダファイルには" "を使います。

● 例

```
// 基本的なヘッダファイルのインクルードには<>を使う
#include <iostream>
#include <string>
// プロジェクト特有のヘッダファイルのインクルードには""を使う
#include "myprog.h"
```

06 ローカル変数であることを示す記憶クラス指定子 auto

書式	auto *decl*
引数	*decl* ……宣言。

● 解説

変数の範囲がローカルであることを示します。ブロックスコープ（{から}までの範囲）の変数宣言のデフォルトですから、通常は省略します。

● 例

```
// ローカル変数を宣言する
auto double v;
```

07 定数の定義 const

C++

書式	const *decl* ; *member-func* const ;
パラメータ	*decl* …………………宣言。 *member-func* ………メンバ関数名。

● 解説

変数や関数を変更できないもの（定数）と指定します。定数を型付きで定義できるので、型の厳密なチェックに役立ちます。

クラスにconstを指定した場合、メンバにmutableを指定すると、そのメンバは変更できるようになります。

●例1

「02-01 基本要素」の「定数」を参照してください。

●例2

```
// 文字定数（文字列定数の定義には''を使う）
const char c = 'A';
// 文字列定数（文字列定数の定義には""を使う）
// stringは標準C++のクラス
const string MyName = "Pochi Dog";
// 実数値の定数
const int pi = 3.14;
// 整数値の定数
const int n = 5;
char name[256] = "Pochi";
// ポインタ宣言。定数変数のポインタしか代入できない
char *const pname = name; // 定数ポインタ
```

08 列挙型の定義 enum

書式

enum [*tag*]
{
  id [= *value*]
  [, ...]
}

パラメータ

tag ………列挙型の名前。省略可能ですが、この列挙型の変数を宣言して使いたいときには必須です。

id ………定数（列挙子）の名前。

value ……定数の値。省略可能です（解説参照）。

●解説

列挙型は、一連の整数の定数（列挙子）を定義するときに使います。定数の値valueは省略可能で、その場合、デフォルトでは最初の定数の値は0から始まり、以降の定数には1つ前の定数より1ずつ大きな値が割り当てられます。

●例

次の例は、定数を7種類定義する例です。DATA_IDの実際の値は-1、DATA_NAMEの値は0、DATA_ADDRESS1の値は1、DATA_ADDRESS2の値は2、DATA_UNKNOWNの値は99です。

```
enum
{
    DATA_ID = -1,
    DATA_NAME,
    DATA_ADDRESS1,
    DATA_ADDRESS2,
    DATA_ADDRESS3,
    DATA_PHONE,
    DATA_UNKNOWN = 99
};
```

次の例は、DiskTypeという名前で、TYPE_FD（値は0）、TYPE_HD（1）、TYPE_MO（2）、TYPE_CDROM（3）、TYPE_UNKNOWN（4）という5個の定数を定義し、DiskType型の変数を宣言する例です。

```
enum DiskType
{
    TYPE_FD,
    TYPE_HD,
    TYPE_MO,
    TYPE_CDROM,
    TYPE_UNKNOWN
};

// 列挙型の変数を宣言する
enum DiskType dt; // Cではenumとタグを使って宣言
DiskType dt; // C++ ではタグだけで宣言可能
```

C++ クラスの中でも列挙型を定義できます。ほかのクラスで定義した列挙子にアクセスするには、型あるいは定数名（列挙子名）をクラス名で修飾します。

```
#include <stdlib.h>  // atoi()を使っているため
#include <iostream>

using namespace std;

class DiscClass
{
public: // クラス外からアクセスできるようにする
   enum DiskType
   {
      TYPE_FD = 0x12,
      TYPE_HD,
      TYPE_MO,
      TYPE_CDROM,
      TYPE_UNKNOWN = 99
   };
};

DiscClass::DiskType dt; // C++ ではタグだけで宣言可能

int main(int argc, char* argv[])
{
   if (atoi(argv[1]) == DiscClass::TYPE_FD)
      cout << "TYPE_FD¥n";
   else
      cout << "FD以外¥n";

   return 0;
}
```

09 暗黙の型変換が行われないようにするための指定子 C++ explicit

書式	explicit *decl*
パラメータ	*decl* ……宣言。

● 解説

引数が1個のコンストラクタで、暗黙の型変換が行われないようにしたいときに指定します。

●例

次の例は、2種類のコンストラクタを宣言した例です。

```
#include <iostream>

using namespace std;

class myStorage {
private:
   int val;
public:
   explicit myStorage(int);     // 引数はint
   explicit myStorage(double); // 引数はdouble
   int getVal() { return val; }
};

myStorage::myStorage(int v)
{
   val = v;
}

myStorage::myStorage(double v)
{
   val = (int)v;
}

int main(int argc, char* argv[])
{
   myStorage msi(2);
   myStorage msd(3.14);

   cout << "Val(int)=" << msi.getVal() << endl;
   cout << "Val(double)=" << msd.getVal() << endl;

   return 0;
}
```

10 外部シンボルであることを示す記憶クラス指定子 extern

書式
① extern *decl*
② extern *str-literal decl* [C++]
③ extern *str-literal* { *decl-list* }

パラメータ
decl …………宣言。たとえば、変数や関数の宣言などを記述します。
str-literal……プログラミング言語を指定する指定子。
たとえば、"C"や"C++"など。
decl-list ……宣言リスト。

● 解説

変数または関数が、外部変数または外部関数であることを示します。つまり、その変数や関数の名前（シンボル）が定義されているファイル以外のファイルから、そのシンボルにアクセスできるようにします。変数または関数の実体は、別のソースファイルで定義しても、同一ファイルで後で定義してもかまいません。

externを使って宣言した変数や関数は、ほかのソースファイルから参照することができます。

変数の場合、externを指定すると変数の寿命が静的になり、プログラムの開始時からプログラムの終了時まで有効になります。

● 例1

書式①は、次のような形式で主にほかのモジュールに定義がある変数にアクセスするときに使います。

```
// abc.c
extern int AppName; // AppNameはほかのモジュールで定義されている
// xyz.c
int AppName; // このAppNameにほかのモジュールからアクセスできる
```

[C++] C++では、*str-literal*にプログラミング言語を表わす文字列を指定して、ほかの言語のモジュールとリンクできるようにします。現時点で主なコンパイラでサポートされている言語指定子は"C++"と"C"だけで、C++の場合のデフォルトは"C++"です。

● 注意

*str-literal*の"C"や"C++"は、大文字小文字を識別します。小文字でextern "c"とすると、コンパイル時にエラーになります。

C++でC言語の関数や変数を使うときには、関数や変数の宣言でextern "C"を使います。

02-02 ディレクティブと宣言

●例2

次の例はC++プログラムの関数myfunc()を、C言語のモジュールでも利用できるようにする例です。

```
extern "C" int myfunc();
```

次の例は、C言語のソースに含まれる関数cfunc()に、C++のプログラムからアクセスするためにextern "C"を使う例です。

```
//
// main.cpp C++のソース
//
#include <iostream>
// int cfunc(int v);ではシンボルが解決しない

extern "C" int cfunc(int v);

using namespace std;

void main()
{
   cout << cfunc(2) << endl;
}



//
// cfunc.c C言語ソース
//
int cfunc(int v)
{
   return v * 2;
}
```

11 値を変更できるようにする記憶クラス指定子 mutable

C++

書式	mutable *decl*;
パラメータ	*decl* ……メンバ変数の宣言。

● 解説

constを指定して定数として作成した関数やconstを指定したオブジェクトのメンバ変数の値を変更できるようにしたいときに、そのメンバ変数に指定する記憶クラス指定子です。

● 例

次の例では、valにはコンストラクタで2が代入されますが、increment()でインクリメントされて出力される結果はVal=3になります。

```
#include <iostream>
using namespace std;

class myStorage {
private:
   mutable int val;
public:
   myStorage(int);
   int getVal() const { return val; }
   void increment() const { val++; }
};

myStorage::myStorage(int v)
{
   val = v;
}

int main(int argc, char* argv[])
{
   const myStorage ms(2);
   ms.increment();
   cout << "Val=" << ms.getVal() << endl;
   return 0;
}
```

12 名前空間の宣言 namespace

C++

書式	namespace *name* { *member* }
パラメータ	*name* ………名前空間の名前。 *member* ……名前空間のメンバの名前。

●解説

名前空間で識別できる名前空間に、変数や関数名などを宣言します。

たとえば、次のnamespace宣言は、dogsという名前空間に、関数add11()を定義します。

●例

```
#include <iostream>
using namespace std;

namespace dogs {
   int add11(int n) {return n + 11;}
}

void main(int argc, char* argv[])
{
   cout << dogs::add11(2) << endl;
}
```

C++で標準入出力ストリームを使う場合には、ほとんど常に次のようにusing namespace std;を使います。

```
#include <iostream>
using namespace std;

int main()
{
   cout << "こんにちワンワン" << endl;
   return 0;
}
```

●注意

コンパイラによっては名前空間をサポートしていないことがあります。

●参照→using

13 レジスタ変数であることを示す記憶クラス指定子 register

書式	register *decl*
パラメータ	*decl* ……宣言。

解説

変数の値をCPUのレジスタに保存するようなコードを生成することをコンパイラに指示します。通常、CPUが変数の値を操作するときには、変数の値をメモリからレジスタにロードして操作し、再びメモリに戻すという作業を行いますが、レジスタに保存した値はCPUが直接操作できるので、素早く実行されることが期待されます。ただし、CPUのレジスタには限りがあるので、register宣言したからといって、処理が必ず早くなるとは限りません。一般的にいって、ループで速度に非常に重大な影響を及ぼす重要なローカル変数に使うと効果がある場合があります。

関数をregister宣言することもありますが、効果は条件によって異なります。

注意

実際に生成されるコードはコンパイラとコンパイル時のオプションに依存します。また、利用できるCPUのレジスタは限りがあるため、register宣言しても値がレジスタに保存されるとは限りません。

例

次の例は、ループのカウンタ変数iとループの中で使っている変数vをレジスタ変数にするように指定します。registerを指定することで、単にint i, v;とするよりプログラムの実行にかかる時間が短くなることが期待できます。

```
register int i, v;

v = 0;

for (i=0; i<30000;i++)
   v = v + i;
```

14 静的なシンボルであることを示す記憶クラス指定子 static

書式	static *decl*
パラメータ	*decl* ……宣言。

● 解説

変数やメソッドが静的（スタティック）であることを示します。

関数の中で静的に宣言されたローカル変数は、関数が実行されて終了しても破棄されません。その関数が次に呼び出されたときには、以前に関数を実行したときの最後の値が入っています。

static宣言した変数や関数・メソッドは1つだけ作成され、モジュールやクラスのすべてのオブジェクトがその同じオブジェクトを参照します。クラスのインスタンスをいくつ生成しても、static関数・メソッドとstatic変数は1つしか作成されません。

● 例

次の例はグローバルな静的変数svと関数の中の静的変数kを宣言する例です。このように静的に宣言した変数の初期化は、変数が作成されたときに1回だけ行われます。そのため、kの値はfunc()が呼び出されるたびに増えていきます。

```
#include <stdio.h>
#include <stdlib.h>

// 静的変数を宣言する
static int sv = 0;

int func(int n)
{
   static k;
   k += n;
   return k;
}

int main(int argc, char* argv[])
{
   sv = atoi(argv[1]);

   printf("1回目sv=%d k=%d¥n", sv++, func(1));
   printf("2回目sv=%d k=%d¥n", sv++, func(1));
   printf("3回目sv=%d k=%d¥n", sv,  func(1));
   return 0;
}
```

15 構造体を宣言する struct

書式	struct [*tag*] { 　*member-list* } [*decl*];
パラメータ	*tag* ………………構造体を識別するタグ。 *member-list* ……構造体のメンバのリスト。 *decl*………………変数宣言。

解説

構造体を宣言します。構造体は、いくつかのメンバで構成されるデータ構造です。構造体のことを「型」あるいは「データ型」、「ユーザー定義型」と呼ぶことがあります。

構造体を宣言すると同時に変数を宣言したいときには、*decl*に変数名を列挙します。

例

次の例は点の座標を表す構造体XYを宣言して使う例です。

```
#include <stdio.h>
#include <memory.h>

struct XY //構造体の宣言
{
   int x;
   int y;
} b;

typedef struct XY Point; //構造体をデータ型として宣言

int main(int argc, char *argv[])
{
   Point a; //構造体の変数を宣言

   memset(&a, 0, sizeof(a));
   b.x = 10;
   b.y = 20;
   a = b;
   printf("%d, %d¥n", a.x, a.y);

   return 0;
}
```

16 名前の定義 typedef

書式	① typedef *type-decl synonym* ; ② typedef struct *struct-tag* { 　(*struct definition*) } *structsynonym*;
パラメータ	*type-decl* ……………定義する型。 *synonym* ……………定義した名前。 *struct-tag* ……………構造体の名前（タグ）。 *struct-def* ……………構造体定義。 *structsynonym* ………構造体を型として定義したときの型の名前。

●解説

すでに定義されている型や構造体と同じ意味を持つ、別の名前を定義します。たとえば、typedef char * PSTR;はchar *をPSTRとして定義します。すると、それ以降、char *と記述する代わりにPSTRを使うことができます。

●例

```
typedef char * PSTR; // char *をPSTR型として定義する

// nameとaddressという要素を持つ構造体custom_tagを
// 定義して、それをCUSTOMER型として定義する
typedef struct custom_tag
{
   PSTR name;
   PSTR address;
} CUSTOMER;
// 以降、CUSTOMER c;のようにCUSTOMER型の変数を宣言できる
struct XY
{
   int x;
   int y;
} b;

// 構造体XYをPointという名前で定義する
typedef struct XY Point;

// unsigned intをUINTとして定義する
typedef unsigned int UINT;
```

17 型の異なる値を保存する構造を作る union

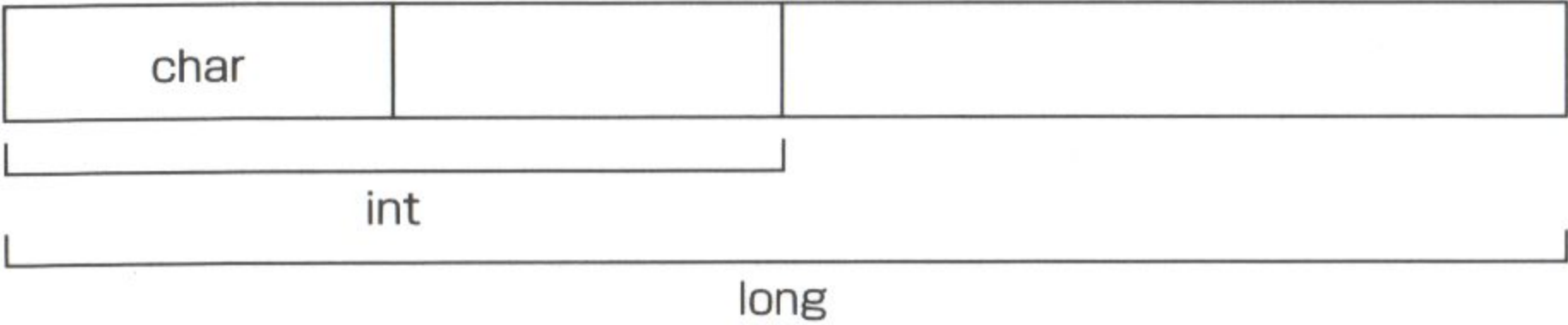
書式	union [*tag*] { *member-list* } [*decl*];
パラメータ	*tag* ………………共用体を識別するタグ。 *member-list* ……共用体のメンバのリスト。 *decl*………………変数宣言。

解説

共用体は型の異なる値を保存するときに使います。構造体とは違って、共用体ではどのメンバにも同じ値が保存され、それぞれの型の値として解釈されます。

たとえば、char、int、longの共用体のメモリ上の状態は次のようになります（charは8ビット、intは16ビット、longは32ビットと仮定します）。

char		

int

long

例

```
// 共用体を宣言する
union AINT {
   long l;
   int i;
   char ch;
} v;

int main(int argc, char* argv[])
{
   int n;

   scanf("%d", &n);
   // v に整数として保存する
   v.i = n;

   // 文字として取り出す
   printf("%c¥n", v.ch);

   // longとして取り出す
   printf("%ld¥n", v.l);

   return 0;
}
```

18 符号なしとして宣言する unsigned

書式	unsigned *decl;*
パラメータ	*decl* ……宣言。

●解説

符号なしの型として宣言するときに、型の名前の一部として使います。たとえば、unsigned charは符号なしの文字型、unsigned intは符号なしのint型です。

●例

```
// unsigned intをUINTとして定義する
typedef unsigned int UINT;
```

unsignedを付けて明示的に符合なしとして宣言する組み込み型には、次のようなものがあります。

```
unsigned char
unsigned short
unsigned int
unsigned long
```

一方、次のような型は、一般にunsignedを付けなくても符号なしとして扱われます。

```
BYTE
DWORD
UINT
WORD
```

多くのコンパイラは、オプションを適切に設定すると、charをunsigned charとして扱うことができます。

●備考

明示的に符合付きとして宣言したいときには、signedを付けます。たとえば、符号付き文字を宣言するときには、signed char c;のようにします。

19 使用する名前空間の指定 using

C++

書式	① using namespace *space-name*; ② using *space-name* :: *member-name* ;
パラメータ	*space-name* ………名前空間の名前。 *member-name* ……名前空間を指定するメンバ名。

解説

プログラムで使用する名前空間を指定します。

名前空間を使うと、同じ名前の変数や関数を異なる用途に定義したり使ったりすることができます。そのため、既存のモジュールやライブラリで同じ名前の関数や変数がすでに定義されている場合でも、同じ名前で別の変数や関数を定義できます。

using namespaceの最も一般的な使い方は、ソースプログラムの先頭のほうで、そこで使う名前空間を次のように指定する方法です。

```
using namespace std;
```

これで、以降、名前空間stdの識別子を修飾なしで使うことができます。

例

```
#include <iostream>

using namespace std;

void main(int argc, char* argv[])
{
   cout << "Hello World!" << endl;
}
```

備考

標準C++ライブラリを使うときには、ほとんど常にusing namespace std;を使います。上の例でusing namespace std;の行がない場合は、名前空間stdで定義されている識別子（たとえば、coutやendl）にstd::を付けなければなりません。

```
std::cout << "Hello World!" << std::endl;
```

あるいは、次のように2番目の書式using space-name :: member-name ;を使って、メンバ名と名前空間を指定します。

```
void main(int argc, char* argv[])
{
   using std::cout;
   using std::endl;
   cout << "Hello World!" << endl;
}
```


20 型または引数がないことを示す void

書式	void [*decl*;]
パラメータ	*decl* ……宣言。

● 解説

関数の型として指定したときには、その関数が戻り値を返さないことを意味します。

関数の引数リストにvoidを指定すると、その関数には引数（パラメータ）がないことを意味します。

void *pの形式で、型の不明なポインタ変数pを宣言することができます。

● 例

次の例は、戻り値も引数もない関数の例です。

```
void DispHello(void)
{
   printf("Hello World!¥n");
}
```

21 変更可能なオブジェクトであることを示す volatile

書式	volatile *decl*
パラメータ	*decl* ……宣言。

◉解説

宣言*decl*の値を、プログラムの外部から変更可能であることを示します。たとえば、割り込みルーチンや入出力ポートがvolatileの値を変更することができます。

◉例

```
int volatile v;
volatile int count;
```

ステートメント

ここでは、実行の制御に使われるC言語とC++の以下のステートメントについて説明します。

項目	機能
break	ループを中断する
case	条件に応じて実行するコードを選択する
continue	繰り返しの先頭に戻る
default	switch～case文のデフォルトの動作を指定する
do	プログラムコードを繰り返す
for	プログラムコードを繰り返す
goto	指定したラベルにジャンプする
if	式を評価して実行するステートメントを決定する
return	呼び出し側の関数に制御を戻す
switch	条件に応じて実行するステートメントを切り替える
while	プログラムコードを繰り返す

01 制御の流れを中断する break

書式　break;

解説

一番内側のdo、for、switch、whileステートメントのいずれかを終了させます。また、switchの中のブロックであるcaseを終了するときにも使います。

breakは、ループまたはswitch～caseブロックで使います。

例

次の例はnが-1になったときにdoループを抜け出ます。

```
int n = 10;

do {
   if (n == -1)
      break; // nが-1になったらループを抜け出る
   n -= 1;
} while (i < n);
```

参照→switch

02 条件に応じて実行するコードを選択する case

書式　switchの書式参照。

解説

caseは、switchステートメントの中で使い、条件に応じて実行するコードを切り替えます。

参照→switch

03 繰り返しの先頭に戻る continue

書式 continue;

● 解説

doステートメント、forステートメント、whileステートメントのループの中で使って、以降のコードを実行しないで、繰り返しの開始位置に制御を移します。

● 例

次の例では、標準入力から読み込んだ文字が空白文字であるときには、以降のコードを飛ばして、ループの先頭に戻り、次の文字を読み込みます。

```
int c;
while ((c = fgetc(stdin)) != EOF)
{
   // 空白文字なら、スキップする
   if (isspace(c))
      continue;
   // cを操作するための一連のコード
}
```

04 switch～case文のデフォルトの動作を指定する default

書式 default:

● 解説

defaultはswitch～case文で使い、どのcaseにも一致しないときに実行するデフォルトのコードを記述します。

● 参照→switch

05 プログラムコードを繰り返す do

書式	do *statement* while (*expr*) ;
パラメータ	*statement* ……繰り返して実行するステートメント。 *expr* …………繰り返しを継続する条件式。この式の値が真である限り、ステートメントを繰り返し実行します。

●解説

do〜whileは、通常、繰り返しのコードを実行したあとで条件式を評価したいときに使います。

●例

一連のプログラムコードをn回繰り返すときには、次のようにします。

```
do {
    (繰り返すコード)
} while (i < n);
```

doループは、繰り返すコードを必ず1回は実行します。それに対して、forやwhileは、ループの中のコードをたとえ1回も実行しなくても、繰り返しの条件を満足しなければループを終了して、ループの次のコードを実行します。

doループを使った関数の例HRK()と、whileループを使った関数の例HitReturnKey()を以下に示します。

```
void HRK()
{
   char c;
   do
   {
      c = getc(stdin);
   }
   while( c != '¥n' );
}

void HitReturnKey()
{
   char c;
   while((c = getc(stdin)) != '¥n' );
   {
      putchar('¥a'); // ビープを鳴らす
   }
}
```

06 プログラムコードを繰り返す for

書式	for ([*init-expr*]; [*cond-expr*]; [*loop-expr*]) 　*statement*
パラメータ	*init-expr* …………初期化式。ループを開始する前に実行したい式。 *cond-expr* ………ループを継続する条件式。この値が真である限り、ループを継続します。 *loop-expr* ………ループ式。繰り返しごとに評価する式。 *statement* ………繰り返して実行するステートメント。

●解説

forループは、条件式*cond-expr*を満たすまで、ステートメントまたは{ }で囲んだステートメントブロックを繰り返し実行します。

forは、通常、繰り返しの回数があらかじめわかっているか、繰り返すステートメントの中でカウンタ変数が必要であるときに使います。たとえば、n回繰り返すときには、次のようなコードを実行します。

```
for (i = 0; i < n; i++)
{
    (繰り返すコード)
}
```

上のコードで、i = 0が初期化式、i < nが条件式、i++が増分の式です。繰り返すコードが1つの式である場合は{ }を省略できますが、わかりやすいソースコードを書きたいときや単純なミスを防ぐためには、なるべく省略しない方が良いでしょう。

初期化式やループ式は、カンマで区切って複数の式を記述できます。

```
for (i=0, j=0; i<n; i++, j++)
```

C++ C++の場合は、初期化式で変数を宣言することもできます。

```
for (int i=0; i<n; i++)
```

● **例**

次の例は、0から99までの数を加算します。

```
int i, x;

for (i=0, x=0; i<100; i++)
{
   x += i;
}

printf("%d¥n", x);
```

07	ジャンプする goto
書式	goto *name;*
パラメータ	*name* ……ジャンプ先のラベル。

● **解説**

ジャンプ先*name*にジャンプします。ジャンプ先のラベルはコロン（:）を使って定義します。

● **例**

次の例は、iを2で割った余りが1であるとき、ラベルnewlineにジャンプすることで、偶数だけを出力するコードの例です。

```
for (int i=0; i<10; i++)
{
   if (i % 2)
      goto newline;
   cout << i ;
newline:
   cout << endl;
}
```

● **注意**

gotoをむやみに使うとプログラムの流れを追跡しにくくなる可能性があります。ほかに方法がない場合や特別な理由がない限り、gotoは使わないでください。

一見、gotoを使わなければ記述できそうにないコードでも、gotoを使わずに簡潔に記述できることがよくあります。gotoを使わないで記述するときには、次のようなテクニックを使います。

- 複数のステートメントを{ }の中に記述する。
- 一部のコードを別の関数にして、その関数を呼び出す。
- returnを使って関数の途中からリターンする。
- continueを使ってループの残りの部分をスキップする。
- switchステートメントを活用する。

08 式を評価して実行するステートメントを決定する if

書式	if (*expr*) 　*stat1* [else *stat2*]
パラメータ	*expr* ……条件式。この値が真であるとき、*stat1*を実行します。 *stat1*……条件式が真（true）のときに実行するステートメント。 *stat2*……条件式が偽（false）のときに実行するステートメント。

● 解説

かっこで囲まれた式*expr*を評価して、式の値が0以外であれば*stat1*を実行し、そうでなければ*stat2*を実行します。

条件式を評価した結果に応じて実行するコードを変えたいときには、一般にifを使います。これはif～elseの形式で使うこともあります。ステートメントは{ }で囲まれた複数のステートメントでもかまいません。

```
if (i == 8)
    (iが8であるとき実行するコード)

if (i == 8) {
    (iが8であるとき実行する一連のコード)
}

if (i == 8) {
    (iが8であるとき実行する一連のコード)
} else {
    (iが8でないとき実行する一連のコード)
}
```

例

次の例では、関数add()は引数nの値が0より大きい場合はその値に1を加えた値を返し、nが0以下のときは0を返します。

```
int add(int n)
{
   if ( n > 0 )
      return n + 1;
   else
      return 0;
}
```

09 呼び出し側の関数に制御を戻す return

書式	return [*expr*] ;
パラメータ	*expr* ……関数が返す値として評価される式。

解説

関数の実行を終了して、呼び出し側の関数に制御を戻します。式*expr*（値を含む）を指定したときには、returnステートメントは、呼び出し側の関数に値（戻り値）を返します。

関数が返す値は、関数と同じ型でなければなりません。たとえば、関数をint func()のように宣言したときには、返す値はintでなければなりません。float func()のように宣言したときには、返す値は実数値でなければならず、倍精度で計算した値を返すときには(float)でキャストする必要があります。

値を返す必要がない関数であっても、関数が正常終了した（成功）か、何らかのエラーが発生した（失敗）かということだけを示すために、intかブール値を返すようにすることがあります。そのようなときには、一般に次のような方法を使います。

- 関数をint型で宣言して0（成功）または-1（失敗）を返す。
- 型BOOL（と必要に応じてTRUEとFALSE）を定義しておき、関数をBOOL型で宣言し、TRUEまたはtrue（成功）かFALSEまたはfalse（成功）を返す（BOOLやTRUEとFALSEは使用するライブラリのヘッダで定義されていることがある）。

関数が成功したときに正の値を結果として返す関数は、関数が失敗したときに返すエラーコードを負の値にすると、関数は結果の値を返すことができ、しかも、呼び出し側では戻り値をチェックすることでエラーの有無を確認できます。

●例

次の例では、関数add()は引数nの値に1を加えた値を返します。

```
int add(int n)
{
   return n + 1;
}
```

●備考

void以外の型を指定した関数やメソッドでreturnを使って値を返さない流れがあると、普通、コンパイル時にC言語の場合には警告として、C++ではエラーとして報告されます。

10 条件に応じて実行するステートメントを切り替える switch

書式

switch (*expr*)
{
case *const-expr* :
  statement;
  [break;]
default :
  statement-def;
}

パラメータ

expr ……………………評価する式。
const-expr……………定数式。
statement……………実行するステートメント。
statement-def ………デフォルトで実行するステートメント。

●解説

式*expr*を評価して、定数式*const-expr*の値が*expr*に一致するステートメントがあれば実行します。一致するステートメントがないときは、defaultのあとのステートメントを実行します。defaultが存在しないときは、switchブロックの次のステートメントを実行します。

*expr*は整数値になる式でなければなりません。

*const-expr*は整数の定数でなければなりません。このときの定数は、-1、0、1、2…のような数値でも、'a'や'¥n'のような値、あるいは整数定数でもかまいません。

●例

```
#include <stdio.h>

int main(int argc, char* argv[])
{
   int n;
   scanf("%d", &n);
   switch (n) {
   case 0:
      printf("Zero¥n");
      break;
   case 1:
      printf("One¥n");
      break;
   default:
      printf("Other¥n");
      break;
   }

   printf("switch...case...done¥n");
   return 0;
}
```

●注意

あるcaseと次のcaseの間にbreakを入れないと、次のcase文のステートメントも実行されてしまいます。たとえば、次の例では、case 0のときに実行するステートメントブロックの最後にbreakがないので、nが0のときにはZeroとOneが出力されます。

●誤りの例

```
switch (n) {
case 0:
   printf("Zero¥n");
case 1:
   printf("One¥n");
   break;
}
```

●備考

switch { ... }の中の最後のcase、またはdefaultのステートメントの最後にはbreak;を入れる必要はありません。しかし、あとでプログラムを変更したときなどに起こしやすい単純なミスを防ぐために、switch { ... }の最後には必要がなくてもbreak;を入れることを習慣にすると良いでしょう。

11 プログラムコードを繰り返す while

書式	while (*expr*) *statement*
パラメータ	*expr* ……………条件式。ループを継続する条件式を記述します。 *statement* ……繰り返して実行するステートメント。

● 解説

ステートメント*statement*を条件式*expr*が真である間（0になるまで）繰り返し実行します。

● 例

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int c;
   while ((c = getc(stdin)) != EOF)
      putc(c, stderr);

   return 0;
}
```

whileの条件式に1のような値を指定することで、条件が常に真になるようにすることがあります。ただし、このようなテクニックを使うときには、ループから確実に抜け出せるように注意を払う必要があります。

```
int func()
{
   int c, n;
   n = 0;
   // xが入力されるまで繰り返す
   while (1) {
      c = fgetc(stdin);
      if (c == 'x')
         break;
      n++;
   }
   return n;
}
```

演算子

演算子は、算術演算や論理演算、その他の演算や比較、変換などに使う1文字または2文字以上のキーワードです。演算子には優先順位があって、1つの式の中に優先順位が異なる演算子がある場合には、優先順位の高い順に評価されます。

ここでは、C/C++の演算子を、優先順位の高い順に説明します。1つの演算子が複数の機能を持ち、機能によって優先順位が異なることがあるので注意してください。

演算子	機能	
::	スコープ解決演算子	C++
::	グローバルスコープ解決演算子	C++
[]	配列添字	
( )	関数呼び出し	
( )	キャスト	
.	メンバ選択演算子	
->	メンバ選択演算子	
++	後置インクリメント演算子	
--	後置デクリメント演算子	
new	オブジェクトの作成	C++
delete	メモリの解放	C++
++	前置インクリメント演算子	
--	前置デクリメント演算子	
*	間接参照演算子	
&	アドレス演算子	
+	プラス演算子	
-	算術否定演算子	
!	論理NOT演算子	
~	ビットごとの補数演算子	
sizeof	サイズ演算子	
typeid()	型名演算子	C++
(type)	型キャスト演算子	
const_cast	型キャスト（変換）演算子	C++
dynamic_cast	型キャスト（変換）演算子	C++
reinterpret_cast	型キャスト（変換）演算子	C++
static_cast	型キャスト（変換）演算子	C++

演算子	説明	
.*	メンバへの適用ポインタ（オブジェクト）	C++
->*	ポインタを介したクラスメンバへの逆参照ポインタ	C++
*	乗算演算子	
/	除算演算子	
%	剰余（モジュロ）演算子	
+	加算演算子	
-	減算演算子	
<<	左シフト演算子	
>>	右シフト演算子	
<	関係演算子（小なり）	
>	関係演算子（大なり）	
<=	関係演算子（以下）	
>=	関係演算子（以上）	
==	等価演算子	
!=	不等価演算子	
&	ビットごとのAND演算子	
^	ビットごとの排他的OR演算子	
|	ビットごとのOR演算子	
&&	論理AND演算子	
||	論理OR演算子	
e1 ? e2 : e3	条件演算子	
=	代入演算子	
*=	乗算代入演算子	
/=	除算代入演算子	
%=	剰余代入演算子	
+=	加算代入演算子	
-=	減算代入演算子	
<<=	左シフト代入演算子	
>>=	右シフト代入演算子	
&=	ビットごとのAND代入演算子	
|=	ビットごとのOR代入演算子	
^=	ビットごとの排他的OR代入演算子	
,	カンマ演算子	

※ C++ はC++だけで使うことができる演算子です。C言語では使えません。

01 スコープ解決演算子 C++

::

書式	*class::member*
パラメータ	*class* …………クラス名。 *member* ……クラスのメンバ。

● 解説

クラス*class*のメンバ*member*を呼び出します。複数のクラスに同じ名前の複数のメンバ関数や変数などがあって、いずれにもアクセス可能なときに、アクセスすべきクラスを指定するために使います。また、クラスの外からクラスのメンバにアクセスするときにも使います。

● 例

```
#include <stdlib.h>
#include <iostream>

using namespace std;

class DiscClass
{
public: // クラス外からアクセスできるようにする
    enum DiskType
    {
        TYPE_FD = 0x12,
        TYPE_HD,
        TYPE_MO,
        TYPE_CDROM,
        TYPE_UNKNOWN = 99
    };
};

DiscClass::DiskType dt; // C++ ではタグだけで宣言可能

int main(int argc, char* argv[])
{
    if (atoi(argv[1]) == DiscClass::TYPE_FD)
        cout << "TYPE_FD¥n";
    else
        cout << "FD以外¥n";
    return 0;
}
```

● **参照→**::（グローバルスコープ解決演算子）の例も参照してください。

02 グローバルスコープ解決演算子

::

書式 ① ::var
② ::func();

● 解説

どのクラスにも属さないグローバル変数varにアクセスするときや、クラスに属さない関数func()を明示的に呼び出すときに使います。

● 例

次の例は、静的関数laugh()、animalクラスのlaugh()、animalクラスから派生したdogクラスのlaugh()があって、それぞれのlaugh()を呼び出す例です。

```
#include <iostream>

using namespace std;

static void laugh()
{
   cout << "Wahaha" << endl;
}

class animal
{
public:
   animal(){};
   ~animal(){};
   static void laugh(){ cout << "Haha" << endl; }
};

class dog : animal
{
public:
   dog(){};
   ~dog(){};
   void laugh(){ cout << "nop" << endl; }
};

int main (int argc, char *argv[])
{
   dog pochi;
   ::laugh();
   animal::laugh();
   pochi.laugh();

   return 0;
}
```

03	配列添字 []
書式	① [ ] ② [*n*]
パラメータ	*n* ………配列のサイズ、または、アクセスする配列要素を識別するインデックス値。

解説

宣言でtype var [n]としたとき、[n]は配列の要素数（配列のサイズ）をn個に指定します。宣言でtype var [ ]としたとき、要素数が不定の配列を宣言します。

var[n]はn番目の要素にアクセスすることを示します。

例

次の例は文字列をchar配列に保存する例です。

```
#include <stdio.h>
#include <string.h>

char s[] = "Hello Dog!";

int main(int argc, char *argv[])
{
   int i, l;
   l = strlen(s);
   for (i=0; i<l; i+=2)
      printf("%c", s[i]);

   return 0;
}
```

次の例は、10個の要素からなる整数配列vを宣言して、そこにユーザーが入力した値を10個保存し、最後にそれらの値を表示する例です。

```
#include <stdio.h>

int v[10];

int main(int argc, char *argv[])
{
    int i;
    for (i=0; i<10; i++)
        scanf("%d", &v[i]);

    for (i=0; i<10; i++)
        printf("v[%d]=%d¥n", i, v[i]);

    return 0;
}
```


04 関数呼び出し ( )

書式	*func*( );
パラメータ	*func* ……関数名。

● 解説

関数を呼び出すときには、関数名のあとに引数を( )で囲って記述します。関数に引数がない場合でも関数名の最後には( )を付けます。

05 キャスト ( )

書式	① (*type*) *value*; ② *type* (*value*); C++
パラメータ	*type* ……型 *value* ……値

● 解説

値*value*を、指定した型*type*の値に変換します。データ型を強制的に変換することを「キャストする」といいます。キャストはたとえば、関数を呼び出すときや変数に値を代入する際に使います。

書式①は(type)を参照してください。書式②はC++で使うことができます。

● 例

```
#include "iostream"

using namespace std;

int main(int argc, char *argv[])
{
    int c;
    cout << "整数値:";
    cin >> c;
    double v = double(c) * 1.2;
    cout << v << endl;

    return 0;
}
```

● 参照→(type)

06 メンバ選択演算子

.

書式	*expr.member*
パラメータ	*expr* ………クラスや構造体の変数。 *member* …クラスや構造体のメンバ。

● 解説

指定したクラスや構造体の中の指定したメンバにアクセスするときに使います。

*expr*はそのオブジェクトの変数でなければなりません。

ポインタ変数のメンバを参照するときには->を使います。

● 例

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>

typedef char * PSTR;
typedef struct custom_tag
{
    PSTR name;
```

```
   PSTR address;
} CUSTOMER;

char s[] = "Pochi Dog";

int main(int argc, char *argv[])
{
   CUSTOMER Cust;
   Cust.name = malloc(sizeof(s));
   strcpy(Cust.name, s);
   printf("name=%s¥n", Cust.name);

   return 0;
}
```

● 参照→-> (メンバ選択演算子)

07 メンバ選択演算子 ->

書式	*expr -> member*
パラメータ	*expr* ………クラスや構造体のポインタ変数。 *member* …クラスや構造体のメンバ。

● 解説

ポインタ変数で指定したクラスや構造体の中の指定したメンバにアクセスするときに使います。

● 例

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>

typedef char * PSTR;

typedef struct custom_tag
{
   PSTR name;
   PSTR address;
} CUSTOMER;

char s[] = "Pochi Dog";
```

```
int main(int argc, char *argv[])
{
   CUSTOMER *pCust;
   pCust = malloc(sizeof(pCust));
   pCust->name = malloc(sizeof(s));
   strcpy(pCust->name, s);
   printf("name=%s¥n", pCust->name);

   return 0;
}
```

● **参照**→.（メンバ選択演算子）

08 後置インクリメント演算子 ++

書式	*expr*++
パラメータ	*expr* ……値をインクリメントする式。

● **解説**

式（変数）の値を評価したあとで、値をインクリメントして1だけ増やします。

i++ はi=i+iまたはi+=1と同じ値になりますが、iが評価されたあとで1だけ値が増えます。

● **備考**… 前置インクリメントより優先順位が高いことに注意してください。

● **例**

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int i;
   i = 0;
   printf("++i=%d¥n", ++i);
   // 前置インクリメント。結果は++i=1
   printf("i++=%d¥n", i++);
   // 後置インクリメント。結果はi++=1
   printf("i=%d¥n", i);
   // 結果はi=2
   return 0;
}
```

● **参照**→++（前置インクリメント演算子）

09 後置デクリメント演算子
--

書式	*expr*--
パラメータ	*expr* ……値をデクリメントする式。

● 解説

式（変数）の値を評価したあとで、値をデクリメントして1だけ減らします。

● 備考

前置デクリメントより優先順位が高いことに注意してください。

● 例

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int i;
   i = 10;
   printf("--i=%d¥n", --i);
   // 前置デクリメント。結果は--i=9
   printf("i--=%d¥n", i--);
   // 後置デクリメント。結果はi--=9
   printf("i=%d¥n", i);
   // 結果はi=8

   return 0;
}
```

● 参照→ --（後置デクリメント演算子）

10 オブジェクトの作成 C++
new

書式	[::] new [*placement*] {*type-name*} [*initializer*]
パラメータ	*placement* ……付加引数の渡し方。 newをオーバーロードするときに指定します。 *type-name* ……作成する型。 *initializer* ………オブジェクトの初期化値。

解説

オブジェクトを作成して、そのオブジェクトのためのメモリを確保し、確保したメモリの先頭を指すポインタを返します。newは、未使用のヒープメモリにsizeof(typename)バイトを確保して指定した型のオブジェクトを作成しようとします。メモリを確保できない場合は、NULLが返されます。

新しく作成されたオブジェクトは、deleteでそのオブジェクトが解放されるか、プログラムが終了するまで有効です。

newの前に::を付けると、グローバルなnewが呼び出されます。

例

```
#include "iostream"
#include "string"

using namespace std;

#define BUFF_LEN 1024

void main(int argc, char* argv[])
{
   char *buff;
   buff = new char[BUFF_LEN];
   if (!buff){
      cout << "メモリを確保できません。" << endl;
      exit (-1);
   }

   cout << "名前は? >";
   cin.getline(buff, BUFF_LEN);
   cout << "名前は" << buff << "です。" << endl;

   delete buff;

   return 0;
}
```

11 メモリの解放 delete

C++

書式	① [::] delete {[]} *expr* ② delete *array*;
パラメータ	*expr* ……メモリを解放するポインタを指す式。 *array* ……配列の名前。

● 解説

newを使って作成したオブジェクトのメモリを解放します。

● 例

newの例参照。

12 前置インクリメント演算子 ++

書式	++*expr*
パラメータ	*expr* ……値をインクリメントする式。

● 解説

式（変数）の値をインクリメントした（1だけ増やした）あとで、値を評価します。

● 備考

++（後置インクリメント）より優先順位が低いことに注意してください。

13 前置デクリメント演算子 --

書式	--*expr*
パラメータ	*expr* ……値をデクリメントする式。

● 解説

式（変数）の値をデクリメントした（1だけ減らした）あとで、値を評価します。

● 備考

後置デクリメントより優先順位が低いことに注意してください。

14 間接参照演算子
*****

書式	* *expr*
パラメータ	*expr* ……名前。一般的には、ポインタ変数名やポインタを返す関数名を指定します。

●解説

ポインタを介して値にアクセスするときに使います。たとえば、int *pv;はintの値のある場所を指すための変数pvを宣言します。pvに保存される値そのものはアドレスであり、intの値は*pvです。ここでいうアドレスはメモリ上の実アドレスとは限りません。

●例

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

char s[] = "Hello, Pochi!";

int main(int argc, char *argv[])
{
   char *p;
   int *pv;
   // sのアドレスをpに代入する
   p = s;
   // int型の値を入れる場所を確保する
   pv = malloc(sizeof(int));
   *pv = 25;
   printf("p=%s¥n", p);
   printf("*pv=%d¥n", *pv);

   return 0;
}
```

●備考

* は乗算演算子としても使えます。乗算演算子として使うときとは優先順位が異なります。

15	アドレス演算子 &
書式	& *expr*
パラメータ	*expr* ……名前。

解説

変数または関数のアドレスを参照します。このときのアドレスは必ずしもメモリ上の物理的な実アドレスであるとは限りません。

例

```
#include <stdio.h>

char s[] = "Hello, Pochi!";

int main(int argc, char *argv[])
{
   int i;
   i = 8;

   // 変数の値とそのポインタ値を表示する
   printf("iの値=%x &iの値=%p¥n", i, &i);

   // 文字列変数の先頭のアドレスから表示する
   printf("s=%s¥n", &s[0]);

   return 0;
}
```

備考

&はビットごとのAND演算子としても使います。

16	プラス演算子 +
書式	*expr1* + *expr2*
パラメータ	*expr1* ……加算される式または値。 *expr2* ……加算する式または値。

解説

演算子の右と左の値を加算します。

● **例**

次の例はaにbを加算して、変数nに代入します。

```
n = a + b;
```

17	算術否定演算子 -
書式	① *expr1* - *expr2* ② - *expr*
パラメータ	*expr1* ……減算される式または値。 *expr2* ……減算する式または値。 *expr* ……負の値を計算する式または値。

● **解説**

演算子の右の値を減算します。また、符号付きの値の符号を反転します。

● **例**

次の例はaからbを減算して変数nに代入します。2行目の式は、nの値の符号を反転し、nが負の数ならnと絶対値が同じ正の値をmに代入し、nが正の数ならnと絶対値が同じ負の値をmに代入します。

```
n = a - b;
m = -n;
```

18	論理NOT演算子 !
書式	!*expr*
パラメータ	*expr* ……論理NOTで評価する式。

● **解説**

式を論理否定します。式全体が0以外と評価された場合にはtrue(1)を返し、そうでなければfalse(0)を返します。

例

```
if ( !e ) {      // (0 == e) と同じ
   // 0 == e のときに実行するコード。
}
```

19 ビットごとの補数演算子

~

書式	*~expr*
パラメータ	*expr* ……ビットごとの補数を計算する式。

解説

各ビットの値を逆にして、ビットごとの補数を計算します。

例

次の例は16進数でf0f5の値と、そのビットごとの補数を計算した値を16進数で表示します。

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   unsigned int i;
   i = 0xf0f5;

   printf("iの値=%x¥n", i);
   printf("~iの値=%x¥n", (~i));

   return 0;
}
```

一般的な32ビットマシンでは、結果としてf0f5とffff0f0aが表示されます（処理系のint型の定義によって結果は異なることがあります）。

備考

C++では、~はデストラクタを作成するときにも使います。

20 サイズ演算子 sizeof

書式	① sizeof (*expr*) ② sizeof (*type*)
パラメータ	*expr* ……サイズを計算する式。 *type* ……サイズを計算する型。

● 解説

オブジェクト、型または式のサイズをバイト単位で取得します。

データ型や構造体のサイズはマシンやライブラリのバージョンなどによって異なることがあるので、メモリを割り当てるような場合にはサイズを数値で直接指定せずにsizeof ()を使って指定するべきです。

● 例

次の例は、charとint型のサイズを表示します。

```
cout << "charは" << sizeof(char) << "バイト" << endl;
cout << "intは" << sizeof(int) << "バイト" << endl;
```

次の例は、intと構造体XYのサイズを表示します。

```
struct XY
{
   int x;
   int y;
};

int main(int argc, char* argv[])
{
   struct XY p;

   printf("Size of int=%d¥n", sizeof(int));
   printf("Size of struct XY=%d¥n", sizeof(p));

   return 0;
}
```

21 型名演算子 typeid()

C++

ヘッダ	typeinfo.h
書式	① typeid(*type*) ② typeid(*expr*)
パラメータ	*type* ……型情報を取得する型。 *expr* ……型情報を取得する式。

● 解説

型と式の情報を実行時に取得します。返されるのはconst type_infoの参照です。

● 例

```
#include <iostream>
#include <typeinfo.h>

using namespace std;

void main(int argc, char* argv[])
{
   char c;
   signed char sc;
   long l;
   long int li;

   if (typeid( c ) == typeid( sc ))
      cout << "char とsigned char は同じ" << endl;
   else
      cout << "char とsigned char は違う" << endl;

   if (typeid( l ) == typeid( li ))
      cout << "long とlong int は同じ" << endl;
   else
      cout << "long とlong int は違う" << endl;
}
```

実行結果は次のようになります（処理系のデータ型の定義によって結果は異なることがあります）。

```
char とsigned char は違う
long とlong int は同じ
```

22 型キャスト演算子 (type)

書式	(*type*) *expr*
パラメータ	*type* ……強制的に変換する型。 式が評価されたあと、値はこの型になります。 *expr* ……変換する式または値。

解説

型を強制的に変換します。一般的には、型の不一致によるエラーや警告を避けるために使います。

例

次の例は、char型のcをint型に、int型のnをchar型に型変換します。

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    char c = '1';
    int n = 65;

    printf("(int)cの値=%d¥n", (int)c);
    printf("(char)nの値=%c¥n", (char)n);

    return 0;
}
```

結果は次のようになります。

```
(int)cの値=49
(char)nの値=A
```

23 型キャスト（変換）演算子 const_cast　C++

書式	const_cast < *type* > (*expr*)
パラメータ	*type* ……強制的に変換する型。 *expr* ……変換する式または値。

●解説

const、volatile属性を追加したり削除します。

●例

次の例はconst_castを使って定数引数をインクリメントできるようにします。

```
void constfunc(const int *pn)
{
   int *pv;
   pv = const_cast<int *> (pn);
   *pv = (*pn) + 1;    // 引数で渡された定数値が変わる
}
```


24 型キャスト（変換）演算子 dynamic_cast　C++

書式	dynamic_cast < *type* > (*expr*)
パラメータ	*type* ……強制的に変換する型。 *expr* ……変換する式または値。

●解説

*expr*を、指定した型*type*のオブジェクトに変換します。*type*は定義済みクラスまたはvoid*の参照でなければなりません。*expr*は、ポインタまたは参照として評価される式でなければなりません。

キャストが正常にできないと、bad_cast例外が生成されます。

●例

次の例は、dynamic_castを使ってCircleクラスの参照をShapeの参照にキャストする例です。

```
#include <typeinfo>
#include <iostream>

using namespace std;

class Shape {
public:
   virtual void draw() {}
};

class Circle: public Shape {
```

```
public:
   void draw() { cout << "Circle" << endl; };
};

int main(int argc, char *argv[])
{
   Circle* circle = new Circle();
   Circle& ref_circle = *circle;

   try {
      Shape& ref_Shape = dynamic_cast<Shape&>(ref_circle);
      ref_Shape.draw();
   }
   catch (bad_cast) {
      cout << "bad_cast exception" << endl;
   }
}
```

25 型キャスト（変換）演算子 reinterpret_cast

C++

書式	reinterpret_cast < *type* > (*expr*)
パラメータ	*type* ……強制的に変換する型。 *expr* ……変換されるポインタ。

● 解説

ポインタを任意の型に変換します。*type*は変換後の*expr*を十分保存できる大きさでなければなりません。

● 例

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   char c = 'X';

   // 32ビットでコンパイルするものとする
   long l = reinterpret_cast<long>(&c);

   cout << l << endl;
}
```

● **注意**

reinterpret_cast演算子を誤って使うと危険です。可能な場合は、他のキャスト演算子を使うべきです。

26 型キャスト（変換）演算子 static_cast　C++

書式	static_cast < *type* > (*expr*)
パラメータ	*type* ……強制的に変換する型。 *expr* ……変換する式または値。

● **解説**

式を指定した型に変換します。暗黙で実行されるすべての型変換に使うことができます。さらに、どんな値でもvoidにキャストできます。constをconst以外の型にするようなキャストは行えません。

実行時の型チェックは行われないので、変換後の型は保証されません。

● **例**

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   char c = 'X';

   int n = static_cast<int>(c);

   cout << n << endl;
}
```

27 間接参照演算子 .*　C++

書式	*pm-expr* .* *cast-expr*
パラメータ	*pm-expr*………クラス型のオブジェクト。 *cast-expr* ……メンバのポインタ型。

解説

*pm-expr*を*cast-expr*に結合します。

例

```
#include <iostream>

using namespace std;

class MyClass {
public:
   void hello() { cout << "hello" << endl; }
   int m_num;
};

void (MyClass::*pmfn)() = &MyClass::hello;
int MyClass::*pmd = &MyClass::m_num;

int main() {

   MyClass AMyClass;

   // メンバー関数を呼び出す
   (AMyClass.*pmfn)();

   // メンバにアクセスする
   AMyClass.*pmd = 10;

   cout  << AMyClass.*pmd << endl;

}
```

28 ポインタを介したクラスメンバへの逆参照ポインタ C++ ->*

書式	*pm-expr ->* cast-expr*
パラメータ	*pm-expr* ………クラス型のオブジェクトを指すポインタ。 *cast-expr* ………メンバのポインタ型。

解説

*pm-expr*を*cast-expr*に結合します。

● 例

```
#include <iostream>

using namespace std;

class MyClass {
public:
   void hello() { cout << "hello" << endl; }
   int m_num;
};

void (MyClass::*pmfn)() = &MyClass::hello;
int MyClass::*pmd = &MyClass::m_num;

int main() {

   MyClass *pMyClass = new MyClass;

   // メンバ関数にアクセスする
   (pMyClass->*pmfn)();

   // データメンバにアクセスする
   pMyClass->*pmd = 18;

   cout  << pMyClass->*pmd << endl;

   delete pMyClass;
}
```

29 乗算演算子 *

書式	*expr1 * expr2*
パラメータ	*expr1, expr2* ……式または値。

● 解説

式*expr1*の値と式*expr2*の値を掛け合わせます。それぞれの式には、整数値または浮動小数点値を指定できます。それぞれの式の型が一致していなくてもかまいません。乗算した結果、値が大きくなりすぎたり小さくなりすぎたりする可能性がありますから、結果の値の範囲に注意する必要があります。

●例

次の例では、n1にn2を掛けて、結果をnResultに代入します。

```
int n1 = 12, n2 = 24, nResult;
nResult = n1 * n2;
```

30	除算演算子 /
書式	*expr1 / expr2*
パラメータ	*expr1, expr2* ……式または値。

●解説

式*expr1*の値を式*expr2*の値で割ります。式*expr2*の値は0であってはなりません。

それぞれの式には、整数値または浮動小数点値を指定できます。それぞれの式の型が一致していなくてもかまいません。

●注意

*expr2*が0または非常に小さい値である場合、例外が発生して、プログラムが異常終了します。

●例

次の例は、整数の割り算で/を使う例です。

```
#include <stdio.h>

int main(void)
{
   int v1, v2, r = 0;

   printf("値1>");
   scanf("%d", &v1);

   printf("値2>");
   scanf("%d", &v2);

   if (v2 > 0)
```

```
    r = v1 /v2;

  printf("結果=%d¥n", r);

  // 短絡評価でv2=0の場合は割り算は行われない
  if ((v2 > 0) && (r = v1 /v2))
    printf("結果=%d¥n", r);

  return 0;
}
```


31 剰余（モジュロ）演算子 %

書式	*expr1% expr2*
パラメータ	*expr1, expr2* ……式または値。

● 解説

式*expr1*の値を式*expr2*の値で割ったときの余りを計算します。この演算子の2つの式の値は整数値でなければなりません。

● 例

次の例では、n1をn2で割った余りを計算します。結果は1になります。

```
int n1 = 10, n2 = 3, remainder;
remainder = n1 % n2;
```

32 加算演算子 +

書式	*expr1 + expr2*
パラメータ	*expr1, expr2* ……式または値。

● 解説

式*expr1*の値に式*expr2*を加えます。それぞれの式には、整数値または浮動小数点値を指定できます。それぞれの式の型が一致していなくてもかまいません。

例1

```
#include <stdio.h>

void main(int argc, char* argv[])
{
   int i = 20, r;
   double v = 3.13;

   r = i + v;
   printf("r=%d¥n", r);
   // 結果は整数23になる

   return 0;
}
```

C++ C++では+演算子を連結のために定義することがあります。たとえば、標準C++のstringクラスでは+を使って文字列を連結することができます。

例2

```
#include "iostream"
#include "string"

using namespace std;

string s1 = "abc";

int main(int argc, char *argv[])
{
   string s2 = "xyz";
   string s = s1 + s2;
   cout << s << endl;

   return 0;
}
```

33 減算演算子
-

書式	*expr1* - *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

●解説

式*expr1*の値から式*expr2*の値を引きます。それぞれの式には、整数値または浮動小数点値を指定できます。それぞれの式の型が一致していなくてもかまいません。

●例

次の例は実数値から整数を引く例です。

```
int n = 3;
double v1,v2 = 3.14;
v1 = v2-3;
```

34 左シフト演算子
<<

書式	*expr1* << *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

●解説

式*expr1*の値の各ビットを、式*expr2*の値だけ左にシフトします。

●例

次の例はn1の値である8の各ビットを、2だけ左にシフトします。結果は32になります(int型の定義によって結果が異なることがあります)。

```
int n1 = 8;
cout << (n1 << 2) << endl;
```

●備考

C++では<<は出力ストリーム演算子としても使われます。

35 右シフト演算子 >>

書式	*expr1* >> *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

● 解説

式*expr1*の値の各ビットを、式*expr2*の値だけ右にシフトします。

● 例

次の例はn1の値である8の各ビットを、2だけ右にシフトします。結果は2になります（int型の定義によって結果が異なることがあります）。

```
int n1 = 8;
cout << (n1 >> 2) << endl;
```

● 備考

C++では>>は入力ストリーム演算子としても使われます。

36 関係演算子（小なり） <

書式	*expr1*< *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

● 解説

式*expr1*と式*expr2*の大小関係を判定します。式*expr1*の値が式*expr2*の値より小さいとtrueを返します。

● 例

次の関数は、n1とn2を比較して、その結果を出力します。

```
bool IsGreater(int n1, int n2)
{
   if (n1 < n2)
      cout << "n1の方が小さい" << endl;
   else
      cout << "n1の方が大きいか等しい" << endl;
   return (n1 < n2);
}
```

37 関係演算子（大なり）
>

書式	*expr1* > *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

● 解説

式*expr1*と式*expr2*の大小関係を判定します。式*expr1*の値が式*expr2*の値より大きいとtrueを返します。

● 例

<（関係演算子（小なり））の例参照。

38 関係演算子（以下）
<=

書式	*expr1* <= *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

● 解説

式*expr1*と式*expr2*の大小関係を判定します。式*expr1*の値が式*expr2*の値より小さいか等しいとtrueを返します。

● 例

次の関数は、n1とn2を比較して、その結果を出力します。

```
bool IsGreaterEqual(int n1, int n2)
{
    if (n1 <= n2)
        cout << "n1の方が小さいか等しい" << endl;
    else
        cout << "n1の方が大きい" << endl;
    return (n1 <= n2);
}
```

39 関係演算子（以上）
>=

書式	*expr1 >= expr2*
パラメータ	*expr1, expr2* ……式または値。

解説

式*expr1*と式*expr2*の大小関係を判定します。式*expr1*の値が式*expr2*の値より大きいか等しいとtrueを返します。

例

<=（関係演算子（以下））の例参照。

40 等価演算子
==

書式	*expr1 == expr2*
パラメータ	*expr1, expr2* ……式または値。

解説

式*expr1*と式*expr2*の大小関係を判定します。式*expr1*の値と式*expr2*の値が等しいとtrueを返します。

例

次の関数は、2つの整数値n1とn2を比較して、その結果を出力します。

```
bool IsEqual(int n1, int n2)
{
   if (n1 == n2)
      cout << "n1とn2は等しい" << endl;
   else
      cout << "n1とn2は違う" << endl;
   return (n1 == n2);
}
```

注意

C/C++の等価比較は=ではありません。==の代わりに間違えて=を使うと、値が代入されてその結果が評価されてしまいますから注意しなければなりません。次の関数は、式n1=n2でn2の値がn1に代入されるので、n2が0以外の値のとき、常に「n1とn2は等しい」と出力します。

```
int IsEqual(int n1, int n2)
{
   if (n1 = n2)
   {
      printf("n1とn2は等しい¥n");
      return 1;
   }
   else
      printf("n1とn2は違う");
   return 0;
}
```

41 不等価演算子 !=

書式	*expr1* != *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

●解説

式*expr1*と式*expr2*の大小関係を判定します。式*expr1*の値と式*expr2*の値が等しくないときにtrueを返します。

●例

```
in+ n;
if (n != 99)
   // nが99でないとき実行するコード
```

42 ビットごとのAND演算子 &

書式	*expr1* & *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

●解説

ビットごとのAND演算を行います。AND演算は2つの値のどちらにも含まれるビットだけを取り出す演算です。

●例

次の例は、2つの16進数1230と1204をAND演算して、16進数で結果1200を出力します。

```
#include <stdio.h>
void main(int argc, char* argv[])
{
   int n1 = 0x1230;
   int n2 = 0x1204;
   int n3;
   n3 = n1 & n2;
   printf("%x¥n", n3);

   return 0;
}
```

43 ビットごとの排他的OR演算子 ^

書式	*expr1^expr2*
パラメータ	*expr1, expr2* ……式または値。

●解説

ビットごとの排他的OR演算を行います。排他的OR演算は2つの値のどちらか一方に含まれるビットだけを取り出す演算です。

●例

次の例は、2つの16進数1230と1204を排他的OR演算して、16進数で結果34を出力します。

```
#include <stdio.h>

void main(int argc, char* argv[])
{
   int n1 = 0x1230;
   int n2 = 0x1204;
   int n3;
   n3 = n1 ^ n2;
   printf("%x¥n", n3);

   return 0;
}
```

1	2	3	0

XOR

1	2	0	4

=

0	0	3	4

44 ビットごとのORを演算子

|

書式	*expr1*	*expr2*
パラメータ	*expr1*, *expr2* ……式または値。

● 解説

ビットごとのOR演算を行います。OR演算は2つの値のどちらかに含まれるビットを取り出す演算です。

● 例

次の例は、2つの16進数1230と1204をOR演算して、16進数で結果1234を出力します。

```
#include <stdio.h>
void main(int argc, char* argv[])
{
   int n1 = 0x1230;
   int n2 = 0x1204;
   int n3;
   n3 = n1 | n2;
   printf("%x¥n", n3);

   return 0;
}
```

1	2	3	0

OR

1	2	0	4

=

1	2	3	4

45 論理AND演算子 &&

書式	*expr1* && *expr2*
パラメータ	*expr1*, *expr2* ……式または値。

解説

論理AND演算を行います。論理AND演算は、どちらの式も真（true）のときは真になり、そうでないときには偽（false）になります。

例

次の例は、2つの値がどちらも0より大きいかどうか判断して、結果を出力します。

```
#include <stdio.h>

void main(int argc, char* argv[])
{
    int n1, n2;
    scanf("%d %d", &n1, &n2);
    if ((n1 > 0) && (n2 > 0)) // [*]
        printf("n1もn2も0より大きい¥n");
    else
        printf("n1かn2のどちらかが0以下¥n");

    return 0;
}
```

上の例の[*]が付いた行は、比較演算子のほうが優先順位が高いので、次のようにしてかまいません。

```
if ( n1 > 0 && n2 > 0 )
```

しかしコードがわかりにくくなりそうな場合は、左右の式（*expr1*と*expr2*）をそれぞれかっこで囲むほうが良いでしょう。

46 論理OR演算子 ||

書式	*expr1* || *expr2*
パラメータ	*expr1, expr2* ……式または値。

● 解説

論理OR演算を行います。論理OR演算は、どちらかの式が真（true）のときに真（true）になります。

● 例

次の例は、2つの値がどちらも0より大きいかどうか判断して、結果を出力します。

```
#include <stdio.h>

void main(int argc, char* argv[])
{
   int n1, n2;
   scanf("%d %d", &n1, &n2);
   if ((n1 > 0) || (n2 > 0))
      printf("n1かn2のどちらかが0より大きい¥n");
   else
      printf("n1とn2のどちらも0以下¥n");

   return 0;
}
```

● **参照**→&&（論理AND演算子）

47 条件演算子 e1 ? e2 : e3

書式	*expr1* ? *expr2* : *expr3*
パラメータ	*expr1* …………………式または値。 *expr2, expr3* ………式またはステートメント。

● 解説

式*expr1*の評価結果に従って、式*expr1*の値がtrueなら式*expr2*が評価（実行）され、falseなら式*expr3*が評価（実行）されます。

*expr2*と*expr3*がステートメントであるときは、次のif～else構文と同じです。

```
if (expr1)
   expr2;
else
   expr3;
```

例

次の例では、nが0より大きいときは関数good()が呼び出されて「Good!」と出力され、そうでなければnは-9999になります。

```
#include <stdio.h>

void good()
{
   printf("Good!¥n");
}

int main(int argc, char* argv[])
{
   int n;
   scanf("%d", &n);
   (n > 0) ?  good() : (n = -9999);
   printf("n=%d¥n", n);

   return 0;
}
```

48 代入演算子

=

書式	*var = expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

解説

式の値を変数に代入します。

注意

C/C++では比較には=を単独では使いません。等価比較には==を使います。

49 乗算代入演算子 *=

書式	*var* *= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

●解説

変数の値に、式の値を掛けて、変数に代入します。

●例

次の例では、変数nの値を3倍して、結果をnに代入します。

```
void main(int argc, char* argv[])
{
    int n = 4;
    n *= 3;
    printf("n=%d¥n", n);

    return 0;
}
```

50 除算代入演算子 /=

書式	*var* /= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

●解説

変数の値を、式の値で割って、変数に代入します。

●注意

式*expr*の値は0あるいは、演算した結果オーバーフローを発生するようなきわめて小さい値にしてはなりません。

●例

次の例では、変数n1の値をn2の値で割って、結果をn1に代入します。

```
#include <stdio.h>
#include <math.h>

void main(int argc, char* argv[])
{
   float n1, n2;
   scanf("%f %f", &n1, &n2);
   if (fabs(n2) > 0.0001)
   {
      n1 /= n2;
      printf("n1=%f¥n", n1);
   }
   else
   {
      printf("2番目の数%fが0に近い値です。¥n", n2);
   }
}
```

51	剰余代入演算子 %=
書式	*var* %= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

●解説

変数*var*の値を式*expr*の値で割った余りを変数*var*に代入します。

●注意

式の値は、0であってはなりません。

●例

次の例では、変数n1の値をn2の値で割った余りをn1に代入します。

```
#include <stdio.h>

void main(int argc, char* argv[])
{
   int n1, n2;
   scanf("%d %d", &n1, &n2);
   if (n2 != 0)
   {
      n1 %= n2;
      printf("n1=%d¥n", n1);
   }
   else
   {
      printf("0では割れません¥n");
   }
}
```


52 加算代入演算子 +=

書式	*var* += *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

● 解説

変数の値に、式の値を加算して変数に代入します。

● 例

次の例では、変数nの値に12を加算して結果をnに代入します。

```
void main(int argc, char* argv[])
{
   int n = 2;
   n += 12;
   printf("n=%d¥n", n);

   return 0;
}
```

53	減算代入演算子
	-=
書式	*var* -= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

解説

変数の値に、式の値を減算して変数に代入します。

例

次の例では、変数nの値に、右辺の式（4 * 4）の計算結果を減算して結果をnに代入します。結果として-14が出力されます。

```
void main(int argc, char* argv[])
{
   int n = 2;
   n -= 4 * 4;
   printf("n=%d¥n", n);

   return 0;
}
```

54	左シフト代入演算子
	<<=
書式	*var* <<= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

解説

変数*var*の値の各ビットを、式*expr*の値だけ左にシフトして、結果を変数*var*に代入します。

例

次の例は最初の値n1の各ビットを、第2の値n2だけ左にシフトします。

```
#include <stdio.h>

void main(int argc, char* argv[])
{
   int n1, n2;
   scanf("%d %d", &n1, &n2);
   n1 <<= n2;
   printf("n1=%d¥n", n1);

   return 0;
}
```

たとえば、n1の値が3、n2の値が2のとき、n1には12が代入されます。

55 右シフト代入演算子 >>=

書式	*var* >>= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

●解説

変数の値の各ビットを、式*expr*の値だけ右にシフトして、結果を変数*var*に代入します。

●例

次の例は最初の値n1の各ビットを、第2の値n2だけ右にシフトして、結果を出力します。

```
#include <iostream>

using namespace std;

void main(int argc, char* argv[])
{
   int n1, n2;
   cin >> n1 >> n2;
   cout << (n1 >>= n2) << endl;

   return 0;
}
```

たとえば、n1の値が8で、n2の値が2のとき、n1には2が代入されます。

56 ビットごとのAND代入演算子 &=

書式	*var* &= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

● 解説

変数*var*の値の各ビットを、式*expr*の値でビットごとにAND演算して、結果を変数*var*に代入します。

● 例

次の例は最初の値n1の各ビットを、第2の値n2でビットごとにAND演算して、結果を出力します。

```
#include <stdio.h>
void main(int argc, char* argv[])
{
    int n1, n2;
    scanf("%d %d", &n1, &n2);
    n1 &= n2;
    printf("n1=%x¥n", n1);

    return 0;
}
```

たとえば、n1の値が12、n2の値が7のとき、n1には4が代入されます。

● **参照**→&（ビットごとのAND演算子）

57 ビットごとのOR代入演算子 |=

書式	*var* |= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

● 解説

変数*var*の値の各ビットを、式*expr*の値でビットごとにOR演算して、結果を変数*var*に代入します。

● 例

次の例は最初の値n1の各ビットを、第2の値n2でビットごとにOR演算して、結果を出力します。

```
#include <stdio.h>

void main(int argc, char* argv[])
{
   int n1, n2;
   scanf("%x %x", &n1, &n2);
   n1 |= n2;
   printf("n1=%x¥n", n1);

   return 0;
}
```

たとえば、n1の値が16進数で、a（10進数で10）、n2の値が5のとき、n1には16進数でf（10進数で15）が代入されます。

58　ビットごとの排他的OR代入演算子 ^=

書式	*var* ^= *expr*
パラメータ	*var* ……変数。 *expr* ……式または値。

● 解説

変数*var*の値の各ビットを、式*expr*の値でビットごとに排他的OR演算して、結果を変数*var*に代入します。

● 例

次の例は最初の値n1の各ビットを、第2の値n2でビットごとに排他的OR演算して、結果を出力します。

```
#include <stdio.h>
void main(int argc, char* argv[])
{
   int n1, n2;
   scanf("%x %x", &n1, &n2);
   n1 ^= n2;
   printf("n1=%x¥n", n1);

   return 0;
}
```

たとえば、n1の値が5、n2の値が3のとき、n1には6が代入されます。

● 参照→^（ビットごとの排他的OR演算子）

59 カンマ演算子

,

書式	*expr1,expr2 (,exprn...);*
パラメータ	*expr1, expr2*………式または値。

● 解説

式*expr1*の値を評価したあとで、式*expr2*を評価します。式はいくつ並べてもかまいません。forステートメントの初期化式、条件式、ループ式や、関数の引数などによく使います。

● 例1

```
#include <stdio.h>

void main(int argc, char* argv[])
{
   int n1, n2, i;
   scanf("%d %d", &n1, &n2);
   for (i=0, n2=0; i<n1; i++, n2++) {
      if (i % 2)
      n2++;
   }
   printf("n1=%d n1=%d¥n", n1, n2);

   return 0;
}
```

C++ C++では複数の変数などを宣言すると同時に初期化するときにも、(,カンマ演算子）を使うことができます。

● 例2

```
int k =1, v = 0, c = 'A';
```

例外処理

例外とは、プログラムの実行中に発生する、プログラムの実行に重大な影響を与えるできごとのことです。

例外には、ハードウェア例外とソフトウェア例外があります。ハードウェア例外は、たとえば、0で除算した場合や、数値演算の結果発生するオーバーフローなどがあります。ハードウェア例外は、多くの場合、プログラムを停止させます。ソフトウェア例外は、たとえば、メモリ不足のためにメモリを確保できないときなどに発生します。

例外処理は、ハードウェア、OS、プログラミング言語がそれぞれサポートしています。そのため、例外処理の詳細は、プラットフォームおよび開発ツールによって異なります。また、同じプラットフォーム上で同じ処理系を使っている場合でも、C言語とC++では例外処理が異なります。

原則的には、次の方法で例外を処理します。

- ANSI Cプログラムでは、__except、__finally、__tryを使って構造化例外を記述します。
- C++の例外処理には、try、catch、finally、throwを使います。
  C++では例外処理が言語でサポートされることになっていますが、実装はコンパイラによって異なります。

例外はライブラリ関数でもサポートされています。たとえば、Win32 APIでは、RaiseException()を呼び出すことで例外を生成できます。また、C言語でtry-catchを使えるコンパイラもあります。

ここではC言語とC++の基本的な例外処理のキーワードを取り上げます。

項目	機能
try-catch	例外を捕獲する
__try - __except	例外を捕獲する
try - finally	常に実行するステートメントを指定する
__try - __finally	常に実行するステートメントを指定する
throw	例外を発生させる

01 例外を捕獲する try-catch

書式

try {
  stat1
} catch (*except-decl*) {
  stat2
}

パラメータ

stat1 ……………例外が発生する可能性があるステートメント。
stat2 ……………例外が発生したときに実行するステートメント。
except-decl ……捕獲する例外の宣言。

●解説

tryブロックは、ステートメント*stat1*を試行します。その結果、例外が発生したら、その例外を対応するcatchブロックのステートメント*stat2*で処理します。catchはC++の例外処理のキーワードですから、処理系が独自にサポートしている場合以外は、C言語では使えません。

●例

```
#include <iostream>
#include <string>

using namespace std;

int test()
{
   // 例外を発生させる
   throw;
   return 0;
}

int main(int argc, char* argv[])
{
   // C++ 例外処理
   try
   {
      test();
   }
   catch ( ... ) // 例外ハンドラ
   {
      cout << "C++ の例外が発生しました。¥n";
   }
   return 0;
}
```

02 例外を捕獲する __try - __except

書式

```
__try {
  stat1
}
__except ( filter ) {
  stat2
}
```

パラメータ
stat1……例外が発生する可能性があるステートメント。
stat2……例外が発生したときに実行するステートメント。
filter……捕獲する例外の宣言。

● 解説

__tryブロックで試行して、例外が発生したら__exceptでその例外を処理します。

C言語の構造化例外はC++でも使える場合があります。そのときには、__tryの代わりにtryを使います。

● 例

```
#include <stdio.h>

int test()
{
   char * p = 0;
   *p = 0;
   return 1;
}

int main(int argc, char *argv[])
{
   // Cの構造化例外処理
   __try{
      test();
   } __except( 1 ) {
      printf("構造化例外が生成されました。¥n");
   }
   printf("done¥n");

   return 0;
}
```

03 常に実行するステートメントを指定する try - finally

書式

```
try {
  stat1
} catch (except-decl) {
  stat2
}
finally {
  stat3
}
```

パラメータ
stat1……………例外が発生する可能性があるステートメント。
stat2……………例外が発生したときに実行するステートメント。
stat3……………例外が発生してもしなくても実行するステートメント。
except-decl……捕獲する例外の宣言。

解説

finallyは、try-catch構造のcatchブロックのあとで使って、例外が発生してもしなくても実行するステートメントを記述します。

互換性

ANSI Cでは、__try - __finally を使います。

04 常に実行するステートメントを指定する __try - __finally

書式

```
__try stat1
__finally stat2
```

パラメータ
stat1……例外が発生する可能性があるステートメント。
stat2……例外が発生してもしなくても実行するステートメント。

解説

__finallyは、構造化例外の__tryブロックのあとで使って、例外が発生してもしなくても実行するステートメントを記述します。

参照→try - finally

05 例外を発生させる throw

書式	try {
	  stat1
	} catch (*except-decl*) {
	  stat2
	  throw *except*;
	}

パラメータ	*stat1* ……………………例外が発生する可能性があるステートメント。 *stat2* ……………………例外が発生したときに実行するステートメント。 *except-decl* …………捕獲する例外の宣言。 *except* ………………発生させる例外。

●解説

throwは、try-catch構造のcatchブロックで使って、例外を発生させます。

リソースが使えないなどの理由で何度か試行してもcatchで捕獲した例外を処理できないようなときには、throwを使ってcatchした例外そのものを再スローすることができます。

03：C言語 リファレンス

この章では、標準的なC言語のライブラリに含まれる主要な関数と関連する項目を重点的に説明します。このリファレンス全体に目を通すと、C言語ライブラリ関数の概要を把握することができます。また、目的に応じて必要なところを読んで機能や使い方を調べることができます。

Cライブラリ関数

ここではC言語の標準ライブラリ関数の概要と、種類やその目的について説明します。ここでは、関数の種類ごとに概要を説明しています。

たとえば、文字列を操作する関数を探したいときには「文字と文字列」を、またファイルのデータにアクセスしたいときには「ファイル入出力」のトピックを見てください。個々の関数の詳しい説明は「03-03 関数の説明」で調べてください。

この章ではC言語で使うことができる関数をすべて説明しているわけではありません。使用可能な関数や各関数の詳細はプラットフォームや処理系によって異なる場合があります。また、説明している関数のうち一部のものは、環境によってはライブラリ関数ではなくシステムコールとして実装されています。

ライブラリ関数概要

ライブラリ関数は、C言語のコンパイラと共に提供される基本的な関数ライブラリに含まれる一連の関数です。

これらの関数は、その用途や機能によって、次のような種類に分類することができます。

- ストリーム入出力関数
- ファイルの操作と処理関数
- 文字と文字列関数
- 変換と数値計算関数
- メモリ関数
- 時刻日付管理関数
- プロセスとスレッド関数
- そのほかの関数

ANSI関数とUnicode関数

文字や文字列が引数である関数には、同じ機能で、引数にワイド文字（Unicode）やワイド文字列の値を指定する関数と、ANSI文字の引数を指定する関数が定義されていることがあります。そのような関数では、関数名の一部だけが特定の方法で変えられています。

最も一般的な名前付けの規則は、引数文字（文字列）がワイド文字（Unicode）の関数に対しては、ANSI文字列の関数の先頭にwを付けて、Unicode文字関数とANSI文字関数とを区別する方法です。たとえば、strtod()のワイド文字バージョンの関数はwcstod()で、closedir()のワイド文字バージョンの関数はwclosedir()です。

また、strで始まる文字列操作関数の中には、Unicode関数名をstrの代わりにwcsで始めるものがあります。たとえば、strtol()のワイド文字バージョンの関数はwcstol()で、strtoul()のワイド文字バージョンの関数はwcstoul()です。

標準関数とプラットフォーム/処理系の関数

ANSI Cの関数と同じ名前で、名前の先頭にアンダースコア（_）が付く関数があります。そのような関数は、ISO C++準拠であるか、あるいはWindowsの関数です。たとえば、POSIX関数dup()に対して、ISO C++準拠の_dup()があります。また、ANSI関数stat()のWindowsバージョンは_stat()と_wstat()です。ただし、先頭にアンダースコア（_）がある関数がすべてISO C++準拠かWindowsの関数であるわけではありません。たとえば、_exit()はPOSIX関数です。

また、Windowsでは、セキュリティが強化された関数が用意されている場合があります。そのような関数は関数名のあとに_sが付きます。たとえば、_ftime()のセキュリティが強化されたバージョンは_ftime_s()です。

ストリーム入出力

ストリーム入出力では、データを情報の流れ（ストリーム）として扱い、この連続して流れているとみなされる情報にアクセスします。たとえば、ファイルや標準入出力のような入出力の対象を「ストリーム」と呼びます。たとえば、「ストリームに書き込む」という表現は、ファイルに書き込んだり標準出力（デフォルトではコンソール画面）に出力することを意味します。

ストリーム入出力関数は、ファイルやコンソール（キーボード、ディスプレイ）などに、さまざまなデータを入力したり出力するときに使います。

ストリーム入出力関数はバッファリングされます。つまり、データはデバイスに直接書き込まれたり、デバイスから直接読み込まれるのではなく、バッファと呼ぶ特別な領域に一時的に保存され、そこから取り出されたり、そこに書き込まれます。

入出力関数の一覧表を表3.1に示します。各関数の詳しい説明は、それぞれの関数の解説ページにあります。

03-01　Cライブラリ関数

▼ 表3.1　ストリーム入出力関数

関数	機能	ページ
clearerr()	ファイルのステータスをクリアする	175
fclose()	ファイルを閉じる	201
feof()	ファイルポインタの位置がEOFであるかどうか調べる	201
ferror()	ファイルのエラーをテストする	202
fflush()	ファイルをフラッシュする	203
fgetc()	ファイルから文字を読み出す	203
fgetpos()	ファイルポインタの現在の位置を取得する	204
fgets()	ファイルから文字列を読み出す	206
fileno()	ファイルのデスクリプタを返す	207
_fileno()	ファイルのデスクリプタを返す	207
fopen()	ファイルを開く	210
fprintf()	書式を指定してデータをファイルに書き込む	211
fputc()	文字をファイルに書き込む	211
fputs()	文字列をファイルに書き込む	212
fread()	書式なしデータをファイルから読み出す	213
freopen()	ファイルを開きなおす	214
fscanf()	書式付きデータをファイルから読み出す	215
fseek()	ファイルポインタの位置を移動する	215
fsetpos()	ファイルポインタの位置を設定する	217
ftell()	現在のファイルポインタの位置を取得する	219
fwrite()	データを書式化しないでファイルに書き込む	221
getc()	文字をファイルから読み出す	222
getchar()	文字を標準入力から読み出す	223
gets()	標準入力から1行の文字列を読み出す	230
lseek()	ファイルの読み書きオフセットの位置を変える	244
printf()	書式付きデータを標準出力に書き込む	265
putc()	文字をファイルに書き込む	268
putchar()	文字を標準出力に書き込む	268
puts()	ストリームに1行書き込む	269
rewind()	ファイルポインタの位置をファイルの先頭に移動する	279
scanf()	書式付きデータを標準入力から読み出す	279

setbuf()	ファイルのバッファリングを制御する	281
setvbuf()	ファイルのバッファリングとバッファのサイズを制御する	285
sprintf()	書式付きデータを文字列に書き込む	289
sscanf()	書式付きデータを文字列から読み出す	290
tmpfile()	テンポラリファイルを作成する	310
ungetc()	文字をファイルに戻す	315
vfprintf()	引数リストの値をファイルに出力する	319
vprintf()	引数リストの値を標準出力に出力する	319
vsprintf()	引数リストの値を文字列バッファに出力する	320
wprintf()	ワイド文字を出力する	332

標準入力、標準出力、標準エラー出力

C言語では、標準入力、標準出力、標準エラー出力という3種類の理論上のデバイスを使うことができます。これらのデバイスは、通常、プログラムの開始と共に自動的に開かれて、getc()やputc()などの標準入出力にアクセスする関数で使うことができます。また、関数freopen()を使ってディスクファイルまたはデバイスを開いてアクセスしたり、リダイレクトすることができます（C++には、これに似た機能を提供するものとして、cin、cout、cerrがあります）。

▼ 表3.2 標準入出力

種類	シンボル	デフォルトデバイス
標準入力	stdin	キーボード
標準出力	stdout	ディスプレイ
標準エラー出力	stderr	ディスプレイ

ファイルとディレクトリの操作と処理

ファイルとディレクトリの操作と処理関数は、ファイルやディレクトリを変更したり削除したり、ファイルへのアクセス権を設定したり問い合わせたりするための関数、および低レベルの入出力を行うための関数です。

ファイルやディレクトリの操作と処理のための関数の一覧表を表3.3に示します。各関数の詳しい説明は、それぞれの関数の解説ページにあります。

03-01 Cライブラリ関数

▼ 表3.3 ファイル処理関数

関数	機能	ページ
access()	ファイルのアクセス権をチェックする	150
_access()	ファイルのアクセス権をチェックする	151
chdir()	カレントディレクトリを変更する	169
_chdir()	カレントディレクトリを変更する	170
chmod()	ファイルのアクセスモードを変更する	171
_chmod()	ファイルのアクセスモードを変更する	174
chown()	ファイルの所有者を変更する	174
close()	ファイルデスクリプタを閉じる	176
_close()	ファイルデスクリプタを閉じる	177
closedir()	ディレクトリを閉じる	177
creat()	ファイルかデバイスを作成する	179
_creat()	ファイルかデバイスを作成する	181
dup()	ファイルデスクリプタを複製する	184
_dup()	ファイルデスクリプタを複製する	184
dup2()	ファイルデスクリプタを複製する	185
_dup2()	ファイルデスクリプタを複製する	186
fchmod()	ファイルのモードを変更する	200
fstat()	ファイルの状態を得る	218
_fstat()	ファイルの状態を得る	219
getcwd()	カレントディレクトリ名を取得する	224
_getcwd()	カレントディレクトリ名を取得する	225
isatty()	デスクリプタが端末かどうか調べる	235
_isatty()	デスクリプタが端末かどうか調べる	235
lstat()	ファイルの状態を取得する	244
mkdir()	ディレクトリを作成する	255
_mkdir()	ディレクトリを作成する	255
open()	ファイルやデバイスをオープンする	258
_open()	ファイルやデバイスをオープンする	259
opendir()	ディレクトリを開く	260
read()	ファイルデスクリプタからデータを読み出す	273
_read()	ファイルデスクリプタからデータを読み出す	274

readdir()	ディレクトリを読み込む	275
remove()	ファイルを削除する	277
rename()	ファイル名やディレクトリ名を変更する	278
stat()	ファイルについての情報を取得する	290
_stat()	ファイルについての情報を取得する	291
tmpfile()	テンポラリファイルを作成する	310
tmpnam()	テンポラリファイル名を生成する	311
truncate()	ファイルを指定した長さに切り詰める	314
umask()	ファイル作成マスクをセットする	315
_umask()	ファイル作成マスクをセットする	315
unlink()	名前やファイルを削除する	315
utime()	ファイルのタイムスタンプを設定する	316
utimes()	ファイルのタイムスタンプを変更する	317
_waccess()	ファイルのアクセス権をチェックする	320
_wchdir()	カレントディレクトリを変更する	320
_wchmod()	ファイルのアクセスモードを変更する	321
_wcreat()	ファイルかデバイスを作成する	322
_wgetcwd()	カレントディレクトリ名を取得する	331
_wmkdir()	ディレクトリを作成する	331
_wopen()	ファイルやデバイスをオープンする	332
_wremove()	ファイルを削除する	333
write()	データをファイルに書き込む	333
_write()	データをファイルに書き込む	334
_wstat()	ファイルについての情報を取得する	334

※ファイルシステムの種類に応じて、ファイルの取り扱い方法には違いがあります。
そのため、サポートされている関数や引数に指定する値などが異なることがあります。

03-01 Cライブラリ関数

文字と文字列

文字と文字列操作の関数は、文字の種類を調べたり、文字列をコピーしたり連結する関数です。文字の種類の判別や1文字の大きさは、現在のロケール（locale）に依存します。

文字と文字列関数の一覧表を表3.4に示します。各関数の詳しい説明は、それぞれの関数の解説ページにあります。

▼表3.4　文字と文字列関数

関数	機能	ページ
isalnum()	文字がアルファベットまたは数字であるかどうか調べる	233
isalpha()	文字がアルファベットであるかどうか調べる	234
iscntrl()	文字が制御文字であるかどうか調べる	236
isdigit()	文字が数値のための文字であるかどうか調べる	236
isgraph()	文字がスペースを除いた表示可能な文字であるかどうか調べる	236
islower()	文字が小文字であるかどうか調べる	237
isprint()	文字が表示可能であるかどうか調べる	238
ispunct()	文字が区切り文字であるかどうか調べる	238
isspace()	文字が空白文字であるかどうか調べる	238
isupper()	文字が大文字であるかどうか調べる	239
isxdigit()	文字が16進数の数字文字であるかどうか調べる	239
_mbclen()	マルチバイト文字の長さを取得する	247
_mbctolower()	大文字を小文字にする	247
mblen()	1文字のバイト数を求める	248
_mbspbrk()	文字列に一連の文字のいずれかの文字があるか調べる	248
mbstowcs()	マルチバイト文字列をワイド文字列に変換する	249
mbtowc()	マルチバイト文字をワイド文字に変換する	250
strcat()	2つの文字列を連結する	291
strchr()	文字列中の文字の位置を示す	292
strcmp()	2つの文字列を比較する	292
strcoll()	現在のロケールを使って2つの文字列を比較する	293
strcpy()	文字列をコピーする	294
strcspn()	文字列から特定の文字セットを含まない部分を探す	294
strlen()	文字列の長さを取得する	298
strncat()	2つの文字列を連結する	298

strncmp()	2つの文字列を比較する	298
strncpy()	文字列をコピーする	299
strpbrk()	文字列に一連の文字のいずれかの文字があるか調べる	300
strrchr()	文字列中の文字の位置を後ろから調べる	301
strspn()	文字列から一連の文字を探す	301
strstr()	部分文字列の位置を示す	301
strtod()	ASCII文字列をdouble型の数値に変換する	302
strtok()	文字列から次のトークンを取り出す	303
strxfrm()	ロケール固有情報に基づいて文字列を変換する	307
tolower()	大文字を小文字にする	311
_tolower()	大文字を小文字にする	312
toupper()	小文字を大文字にする	313
_toupper()	小文字を大文字にする	313
towlower()	大文字を小文字にする	313
wclosedir()	ディレクトリを閉じる（ワイド文字バージョン）	321
wcscmp()	ワイド文字の文字列を比較する	322
wcslen()	ワイド文字の文字列の長さを取得する	323
_wcspbrk()	文字列に一連の文字のいずれかの文字があるか調べる	324
wcstod()	ワイド文字列をdouble型の数値に変換する	325
wcstombs()	ワイド文字列をマルチバイト文字列に変換する	326
wctomb()	ワイド文字をマルチバイト文字列に変換する	327

変換と数値計算

変換と数値計算の関数は、数値と文字列を変換したり、さまざまな算術演算を行う関数です。

変換と数値計算関数の一覧表を表3.5に示します。各関数の詳しい説明は、それぞれの関数の解説ページにあります。

03-01　Cライブラリ関数

▼ 表3.5　変換と数値計算関数

関数	機能	ページ
abs()	整数の絶対値を計算する	148
acos()	アークコサインを計算する	153
asin()	アークサインを計算する	156
atan()	アークタンジェントを計算する	158
atan2()	アークタンジェントを計算する	159
atof()	文字列をdouble型の数値に変換する	161
atoi()	文字列をint型に変換する	162
atol()	文字列をロング整数に変換する	163
ceil()	引数の値を繰り上げた数を返す	168
cos()	コサインを計算する	178
cosh()	ハイパーボリックコサインを計算する	178
div()	整数の割り算を行って商と余りを計算する	183
exp()	指数を計算する	199
fabs()	浮動小数点実数の絶対値を計算する	199
floor()	引数の値を越えない最大の整数値を返す	208
fmod()	浮動小数点数の余りを計算する	209
frexp()	浮動小数点実数を仮数と指数に分ける	214
labs()	ロング整数の絶対値を計算する	240
ldexp()	仮数と指数から実数を計算する	240
ldiv()	ロング整数の割り算の商と余りを計算する	241
log()	自然対数を計算する	243
log10()	常用対数を計算する	243
modf()	浮動小数点実数を整数と小数部分に分ける	257
pow()	ベキ乗を計算する	265
sin()	サインを計算する	287
sinh()	ハイパーボリックサインを計算する	287
sqrt()	平方根を計算する	289
strtol()	文字列をロング整数に変換する	304
strtoul()	文字列を符号なしロング整数に変換する	306
tan()	タンジェントを計算する	310
tanh()	ハイパーボリックタンジェントを計算する	310
wcstol()	ワイド文字列をロング整数に変換する	325
wcstoul()	ワイド文字列を符号なしロング整数に変換する	326

メモリ

メモリの管理に関連する関数の一覧表を表3.6に示します。各関数の詳しい説明は、それぞれの関数の解説ページにあります。

▼ 表3.6 メモリ関数

関数	機能	ページ
calloc()	メモリを動的に割り当てる	167
free()	メモリブロックを解放する	213
malloc()	メモリを動的に割り当てる	245
memccpy()	メモリ領域をコピーする	251
memchr()	メモリの中で特定の文字を探す	251
memcmp()	メモリ領域を比較する	253
memcpy()	メモリ領域をコピーする	254
memmove()	メモリ領域をコピーする	255
memset()	メモリ領域を特定のバイトで埋める	255
realloc()	メモリを動的に再割り当てする	275

時刻と日付

時刻と日付の管理や計算、変換などを行う関数の一覧表を表3.7に示します。各関数の詳しい説明は、それぞれの関数の解説ページにあります。

▼ 表3.7 時刻日付管理関数

関数	機能	ページ
asctime()	時刻をASCII文字列に変換する	155
ctime()	時刻のバイナリデータをASCII文字列に変換する	181
difftime()	時刻の間隔の計算	182
ftime()	日付と時間を返す	220
_ftime()	日付と時間を返す	220
gmtime()	カレンダー時刻を万国標準時に変換する	231
localtime()	時刻の値を現地時間に変換する	242
mktime()	現地時刻をカレンダー値に変換する	257
strftime()	日付と時間を書式化する	295
tzset()	時刻の変換情報を初期化する	314
_tzset()	時刻の変換情報を初期化する	314
wcsftime()	日付と時間を書式化する	323
wctime()	時刻のバイナリデータをワイド文字列に変換する	327

03-01 Cライブラリ関数

プロセスと環境の制御

プロセス関数や環境制御の関数は、新しいプロセスを作成してプログラムを起動したり、プログラムを停止したりするときに使います（スレッド関数はシステムに依存するシステムコールなので、本書では取り上げません）。

プロセス関数の一覧表を表3.8に示します。各関数の詳しい説明は、それぞれの関数の解説ページにあります。

▼表3.8 プロセスと環境の関数

関数	機能	ページ
abort()	プログラムの異常終了を発生させる	148
atexit()	プログラムが正常終了した時に呼び出す関数を登録する	160
_cexit()	プロセスを終了する	169
_c_exit()	プロセスを終了する	169
execl()	プログラムを実行する	187
_execl()	プログラムを実行する	188
execle()	プログラムを実行する	189
_execle()	プログラムを実行する	191
execlp()	プログラムを実行する	191
_execlp()	プログラムを実行する	193
execv()	プログラムを実行する	193
_execv()	プログラムを実行する	195
execve()	プログラムを実行する	195
_execve()	プログラムを実行する	196
execvp()	プログラムを実行する	196
_execvp()	プログラムを実行する	197
exit()	プロセスを終了する	197
_exit()	現在のプロセスを終了させる	198
_Exit()	現在のプロセスを終了させる	198
getegid()	グループIDを得る	226
getgid()	グループIDを得る	227
getpid()	プロセスIDを得る	229
_getpid()	プロセスIDを得る	229
getppid()	親プロセスIDを得る	230

_onexit()	プログラムが正常終了した時に呼び出す関数を登録する	257
pause()	シグナルを待つ	261
pclose()	パイプに結合されたファイルを閉じる	262
_pclose()	パイプに結合されたファイルを閉じる	262
pipe()	パイプを作成する	263
_pipe()	パイプを作成する	263
popen()	パイプを作成してコマンドを実行する	263
_popen()	パイプを作成してコマンドを実行する	265
putenv()	環境変数を変更または追加する	269
_putenv()	環境変数を変更または追加する	269
raise()	現在のプロセスにシグナルを送る	270
setenv()	環境変数を設定する	281
signal()	割り込みシグナル処理を設定する	286
swab()	隣接するバイトを交換する	308
_swab()	隣接するバイトを交換する	309
system()	コマンドを実行する	309
unsetenv()	環境変数を削除する	316
_wexecl()	プログラムを実行する	328
_wexecle()	プログラムを実行する	328
_wexeclp()	プログラムを実行する	329
_wexecv()	プログラムを実行する	329
_wexecve()	プログラムを実行する	330
_wexecvp()	プログラムを実行する	330
_wpopen()	パイプを作成してコマンドを実行する	332
_wsystem()	コマンドを実行する	334

そのほかの機能

ライブラリには上記のカテゴリに含まれないさまざまな関数があります。ここに示す関数のいくつかは、特定のプラットフォームにライブラリ関数またはシステムコールとして実装されている関数またはマクロです。ある種の関数には、いくつかの関数を一定の順序で呼び出すことでプログラミングの目的を達成できるものがあります。たとえば、ソケット通信では、socket()を使って生成し、bind()でアドレスに結び付けます。そして、listen()を実行すると接続を待ちます。

03-01 Cライブラリ関数

そのほかの機能に関連する関数の一覧表を表3.9に示します。各関数の詳しい説明は、それぞれの関数の解説ページにあります。

▼ 表3.9 そのほかの関数

関数	機能	ページ
accept()	ソケットへの接続を受け付ける	149
addmntent()	ファイルシステム記述ファイルに情報を追加する	153
assert()	式を評価して、その結果が偽である場合にプログラムを中止する	157
bind()	ソケットに名前を付ける	164
bsearch()	ソートされた配列をバイナリサーチする	165
clock()	プロセッサ時間を返す	175
endmntent()	ファイルシステム記述ファイルを閉じる	186
getenv()	環境変数を取得する	226
getmntent()	ファイルシステム記述ファイルの情報を取得する	228
hasmntopt()	ファイルシステム記述ファイルのオプションを調べる	231
longjmp()	環境を復元する	243
perror()	システムエラーメッセージを出力する	262
qsort()	配列を並べ替える	270
rand()	乱数を生成する	271
random()	乱数を生成する	272
recv()	ソケットからメッセージを受け取る	277
send()	ソケットへメッセージを送る	280
setjmp()	プログラムの現在の環境を保存する	283
setlocale()	現在のロケールを設定する	284
setmntent()	ファイルシステム記述ファイルをオープンする	284
socket()	通信ソケットを作成する	288
srand()	乱数系列を初期化する	289
srandom()	乱数系列を初期化する	290
strerror()	エラーコードを説明する文字列を取得する	294
va_arg()	個数が変わる引数リストを提供する	318
va_end()	個数が変わる引数リストへのアクセスを終了する	318
va_start()	個数が変わる引数リストへのアクセスを開始する	318

グローバル定数と変数

ここではC言語の主な変数と、C言語で使われる主な定数を示します。

定数はプログラムの中で使う値が変化しないシンボルです。定数は、ライブラリのさまざまなヘッダファイルで定義されています。そのため、使うライブラリによって定数の定義の有無や値が異なることがあります。ここでは、C/C++で一般によく使われるグローバルな定義済み定数について説明します。関数特有の定数については、各関数の解説を参照してください。

▼ 表3.10　グローバル定数と変数

項目	機能
BUFSIZ	バッファサイズの定数
EOF	ファイル終端またはエラーを示す定数
errno	システムレベルの関数呼び出しのエラーコードを示す変数
EXIT_FAILURE	処理の失敗を示す定数
EXIT_SUCCESS	処理の成功を示す定数
NULL	値が0であることを示す定数
WEOF	EOFのワイド文字バージョン

以降のページで示す各定数の「定義」は一般的な定義値を表します。実際の定数の値は処理系によって異なることがあるので、ヘッダファイルを調べるか、

```
printf( "定数名=%d ¥n", 定数名 );
```

のような式を使って確認してください。

01 バッファサイズの定数 BUFSIZ

定義	① #define BUFSIZ 512 ② #define BUFSIZ 4096

● **解説**

BUFSIZはバッファのサイズを定義する定数です。実際の定義値は環境によって異なります。

● **関連項目**… setvbuf()

02 ファイル終端またはエラーを示す定数 EOF

定義	#define EOF (-1)

● **解説**

EOF（End Of File）は、入出力関数でファイルの終端に達したときか、エラーが発生したをことを示す定数です。

03 システムレベルの関数呼び出しのエラーコードを示す変数 errno

ヘッダ	errno.h
書式	extern int errno;

● **解説**

システムレベルの関数の呼び出しでエラーが発生すると、errnoにエラーコード（定数）が設定されます。この値は次の関数呼び出しで変更される可能性があるので、エラーコードを調べたいときには関数を実行した直後にerrnoの値をチェックする必要があります。errnoそのものは定数ではなく、システムレベルのグローバル変数です。

errnoの値に対応するエラーメッセージが定義されている場合は、perror()を使ってメッセージを表示したり、strerror()を使ってエラーメッセージ文字列を取得できます。

● **例**… execl()の例参照。

● **関連項目**… perror()、strerror()

04 処理の失敗を示す定数 EXIT_FAILURE

定義	#define EXIT_FAILURE 1

● **解説**

関数で処理や操作が失敗して終了することを示す定数です。関数exit()やatexit()などで返すことがあるほか、必要に応じて処理が失敗したことを示す値として使うことができます。

05 処理の成功を示す定数 EXIT_SUCCESS

定義	#define EXIT_SUCCESS 0

●解説

関数で処理や操作が成功して終了することを示す定数です。関数exit()やatexit()などで返すことがあるほか、必要に応じて処理が成功したことを示す値として使うことができます。

06 値が0であることを示す定数 NULL

定義	① #define NULL 0 ② #define NULL ((void *)0)

●解説

NULLは値0あるいはNULLポインタの値です。

この値は、関数が正常に終了しなかったときに返されることがあります。たとえば、gets()ではエラーが発生するか、ファイルの終端を検出するとNULLポインタを返します。また、長さ0の文字列を表すときにも使います。

●例

次の例は、文字列"12"を作る例です。

```
#include <stdio.h>

void main(int argc, char* argv[])
{
    char str[] = {'1', '2', NULL};

    printf("str=%s¥n", str);

    return 0;
}
```

07 EOFのワイド文字バージョン WEOF

定義	#define WEOF (unsigned short)(0xFFFF)

●解説

入出力関数でファイルの終端に達したときか、エラーが発生したことを示す定数です。これはEOFのワイド文字バージョンです。

●参照→EOF

関数の説明

ここではC言語およびC++のプログラミングで使う主な関数の機能や書式と使用例を示します。

システムがサポートする関数は、プラットフォームと処理系によって違います。また、この章でC言語で使うことができるすべての関数を説明しているわけではありません。

01 プログラムの異常終了を発生させる abort()

ヘッダ	stdlib.h、process.h
書式	void abort(void);

●解説

プログラムを異常終了させます。この関数を呼び出してプログラムを終了すると、すべての開いているストリームは閉じられてフラッシュされます。abort()はシグナルSIGABRTを発生させたいときに使います。通常のプログラムの終了には、関数exit()を呼び出すか、main()関数でreturnステートメントを実行します。

●例

次の例は、ある関数cfunc()が返した値が2を超えたときにプログラムを異常終了する例です。

```
if (cfunc() > 2)
    abort();
```

02 整数の絶対値を計算する abs()

ヘッダ	stdlib.h、math.h
書式	int abs(int *n*);
引数	*n* …………絶対値を求めたい整数。
戻り値	整数の引数の絶対値。

●解説

整数の引数*n*の絶対値を計算します。絶対値は数値から負の符号をとった値で、たとえば、5の絶対値は5、-12の絶対値は12です。

注意

整数の値の範囲はプラットフォームや言語に依存します。

例

次の例は、入力された整数値の絶対値を返します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int n;
    printf("整数値>");
    scanf("%d", &n);
    printf("絶対値は%d ¥n", abs(n));

    return 0;
}
```

参照→fabs()、floor()、labs()

03 ソケットへの接続を受け付ける accept()

ヘッダ	sys/types.h、sys/socket.h
書式	int accept(int *s*, struct sockaddr **addr*, int **addrlen*);
引数	*s* …………ソケットを識別するポインタ。 *addr* ……接続したアドレスを受け取るバッファのポインタ。 *addrlen* …バッファに入るアドレスの長さを保存するための整数のポインタ。
戻り値	成功したときには受け付けられたソケットのデスクリプタ、そうでないときは-1。

解説

ソケットへの接続を受け付けます。一般的には、ソケットはsocket()を使って生成し、bind()でアドレスに結び付けます。そして、その後listen()を呼び出して接続の要求を待ち、accept()を使って接続を受け付けます。

エラーコード

EBADF、EFAULT、ENOTSOCK、EOPNOTSUPP、EWOULDBLOCK

●互換性

ソケットの実装はシステム（主にOS）に強く依存します。エラーコードとして、環境によっては上記のほかに、

EAGAIN、ECONNABORTED、EINTR、EINVAL、EMFILE、ENFILE、ENOBUFS、ENODEV、ENOMEM、ENOSR、EPERM、EPROTO、EPROTONOSUPPORT、ERESTARTSYS、ESOCKTNOSUPPORT、ETIMEDOUT

などを返すことがあります。

04 ファイルのアクセス権をチェックする access()

ヘッダ	unistd.h
書式	int access(const char **path*, int *mode*);
引数	*path*……アクセス権を調べるファイル名。 *mode* …ファイルのモード。解説の表に示す値のいずれか。
戻り値	要求がすべて満たされる場合は0。そうでなければ-1。

●解説

指定したファイルに対して、読み出しや書き込みが許されているか調べます。

*mode*の値は次の表に示すとおりです。

modeの値	意味
R_OK	ファイルが存在し、読み出し許可があるかチェックするように要求する。
W_OK	ファイルが存在し、書き込み許可があるかチェックするように要求する。
X_OK	ファイルが存在し、実行許可があるかチェックするように要求する。
F_OK	ファイルが存在するかどうかをチェックするように要求する。

*path*がシンボリックリンクの場合は、シンボリックリンクが参照するファイルに対する権利がチェックされます。また、ファイルの存在をチェックすることもできます。

この関数で実行可能という結果が得られても、それがほかのシステムのファイルである場合（たとえばUNIX上にDOSの実行可能ファイルがある場合）、実行できない可能性があります。

●エラーコード

EACCES、EFAULT、EINVAL、EIO、ELOOP、ENAMETOOLONG、ENOENT、ENOMEM、ENOTDIR、EROFS

●準拠… POSIX

● 例

次の例は、ユーザーが入力したファイル名のファイルが存在するかどうかと、ファイルが存在する場合、アクセス権を表示します。

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   char buff[BUFSIZ];
   printf("ファイル名>");
   scanf("%s", buff);

   /* 存在チェック */
   if((access(buff, F_OK)) == 0)
   {
      printf("ファイル%s は存在します。¥n", buff);
      /* 書き込みアクセス権のチェック */
      if((access(buff, W_OK)) == 0)
         printf("書き込み可能です。¥n");
      /* 読み出しアクセス権のチェック */
      if((access(buff, R_OK)) == 0)
         printf("読み出し可能です。¥n");
      /* 実行アクセス権のチェック */
      if((access(buff, X_OK)) == 0)
         printf("実行可能です。¥n");
   }
   else
      printf("ファイル%s は存在しません。¥n", buff);
}
```

05 ファイルのアクセス権をチェックする _access()

ヘッダ	io.h
書式	int _access(const char **path*, int *mode*);
引数	*path* ……アクセス権を調べるファイル名。 *mode* …ファイルのモード。解説の表に示す値のいずれか。
戻り値	要求がすべて満たされる場合は0。そうでなければ-1。

● 解説

指定したファイルに対して、読み出しや書き込みが許されているか調べます。

03-03 関数の説明

*mode*の値は次の表に示すとおりです。

modeの値	意味
00	存在するかどうかだけチェックするように要求する。
02	書き込み専用であるかチェックするように要求する。
04	読み取り専用であるかチェックするように要求する。
06	読み書き可能であるかチェックするように要求する。

●例

次の例は、Windows環境で実行するプログラムの例で、ユーザーが入力したファイル名のファイルが存在するかどうかと、ファイルが存在する場合、アクセス権を表示します。

```
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#ifdef WIN32
#define F_OK 0
#define W_OK 2
#define R_OK 4
#define RW_OK 6
#endif

int main(int argc, char *argv[])
{
   char buff[BUFSIZ];
   printf("ファイル名>");
   scanf("%s", buff);

   /* 存在チェック */
   if((_access(buff, F_OK)) == 0)
   {
      printf("ファイル%s は存在します。¥n", buff);
      /* 書き込みアクセス権のチェック */
      if((_access(buff, W_OK)) == 0)
         printf("書き込み可能です。¥n");
      /* 読み出しアクセス権のチェック */
      if((_access(buff, R_OK)) == 0)
         printf("読み出し可能です。¥n");
   }
   else
      printf("ファイル%s は存在しません。¥n", buff);
}
```

● 参照→access()

06	アークコサインを計算する acos()
ヘッダ	math.h
書式	double acos(double *x*);
引数	*x* ………アークコサインを計算したい角度（ラジアン単位）。
戻り値	成功するとアークコサインの値（ラジアン単位）。そうでない場合は、戻り値は不定。

● 解説

指定した角度に対するアークコサイン（逆余弦）を計算します。角度はラジアンで指定し、その値は-1～1の範囲でなければなりません。

● エラーコード… EDOM

● 例

sin()の例参照。

07	ファイルシステム記述ファイルに情報を追加する addmntent()
ヘッダ	stdio.h、mntent.h
書式	int addmntent(FILE **fp*, const struct mntent **mnt*);
引数	*fp* ………ファイルシステム記述ファイルのファイルポインタ。 通常は、setmntent()が返した値。 *mnt* ……追加する情報を入れたmntent構造体のポインタ。
戻り値	成功したときは0、そうでなければ1。

● 解説

指定したファイルシステム記述ファイルの最後にmntent構造体の情報を追加します。
この関数を実行するには、ファイルシステム記述ファイルを操作する権限が必要です。

● 仕様… BSD

● 互換性

この関数はUNIX系OSの記述ファイルを操作するための関数であり、Windowsにはありません。

●例

次の例は、UNIX系OSのファイルシステム記述ファイル/etc/fstabに1行追加します。

```
#include <stdio.h>
#include <mntent.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   FILE *filep;
   struct mntent *pmnt;
   pmnt = malloc(sizeof(struct mntent));
   filep = setmntent("/etc/fstab", "a");
   if (filep == NULL)
   {
      fprintf(stderr, "ファイル/etc/fstabを開けません.¥n");
      exit -1;
   }
   pmnt->mnt_fsname = "/dev/fd0"; /* マウントするファイルシステムの名前 */
   pmnt->mnt_dir = "/dosfd";  /* ファイルシステムのパスプリフィックス */
   pmnt->mnt_type = "vfat";   /* マウントタイプ（mntent.h参照） */
   pmnt->mnt_opts = "noauto,user"; /* マウントオプション（mntent.h参照） */
   pmnt->mnt_freq = 0;        /* ダンプするかどうか */
   pmnt->mnt_passno = 0;      /* ファイルチェックするかどうか */
   if (addmntent(filep, pmnt))
      fprintf(stderr, "addmntent()は失敗しました.¥n");
   endmntent(filep);
   free(pmnt);
}
```

●関連ファイル

/etc/fstab ……ファイルシステム記述ファイル

/etc/mtab ……マウントされたファイルシステム記述ファイル

●参照→fopen()、setmntent()

08	時刻をASCII文字列に変換する asctime()
ヘッダ	time.h
書式	char *asctime(const struct tm *ptime);
引数	*ptime* ……tm構造体のポインタ。
戻り値	時刻を示すASCII文字列。

解説

構造体*ptime*に入っている時間をASCII文字列に変換します。tm構造体は、時刻の情報を時分秒年月日などに分割して保存する構造体で、<time.h>で以下のように定義されています。

```
struct tm
{
   int tm_sec;   /* 秒。0～59の値。 */
   int tm_min;   /* 分。0～59までの値。 */
   int tm_hour; /* 時間。真夜中からの通算時間、0～23までの値。 */
   int tm_mday; /* 日。月はじめからの日数、1から31までの値。 */
   int tm_mon;   /* 月。1月からの通算月数、0～11までの値。 */
   int tm_year; /* 年。1900 年からの通算年数。 */
   int tm_wday; /* 曜日。日曜日からの通算日数(曜日)。0～6までの値。 */
   int tm_yday; /* 1月1日からの通算日数。0～365までの値。 */
   int tm_isdst;/* 夏時間のフラグ。正の値ならば夏時間は有効、0なら無効、
                 * 負の値ならこの情報には意味がない。 */
};
```

tm構造体のtm_secは、通常は0～59の値ですが、閏秒のために61までの値が入れられることがあります。

返される値の形式は次のとおりです。

Wek Mon dd hh:mm:ss yyyy¥n¥0

上の書式で、Wekは週、Monは月、ddは日付、hhは時、mmは分、ssは秒、yyyyは年です。

互換性

この関数の、ANSI文字バージョンはasctime()、ワイド文字バージョンは_wasctime()です。

例

次の例は、現在の時刻（日付と時間）を、現地時間と万国標準時（UTC、世界協定時刻）で表示します。

```
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

struct tm *UTCtime;
time_t utime;

int main(int argc, char *argv[])
{
#ifdef WIN32
   _tzset();
#else
   tzset();
#endif
   /* 現在の日時を取得する */
   time(&utime);
   /* 文字列に変換して表示する */
   printf("現在の日時: %s", ctime(&utime));
   /* 現在の日時（万国標準時、UTC）を秒単位で取得する */
   time(&utime);
   /* 時刻をtm構造体形式の万国標準時に変換する */
   UTCtime = gmtime(&utime);
   printf("万国標準時: %s", asctime(UTCtime));

   return 0;
}
```

09 アークサインを計算する asin()

ヘッダ	math.h
書式	double asin(double *x*);
引数	*x* ………アークサインを計算したい角度（ラジアン単位）。
戻り値	アークサインの値（ラジアン単位）。

● 解説

指定した引数の値のアークサイン（逆正弦）を計算します。

● エラーコード… EDOM

● 例

次の例は、入力された角度に対するアークサインの値を表示します。

```
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ];
   double x;

   printf("角度（ラジアン、-1～1の範囲の実数）= ");
   scanf("%s", buff);
   x = atof(buff);
   if (x < -1 || x > 1)
   {
      puts("角度が範囲外です。");
      return -1;
   }
   printf("%.2fのアークサインの値= %.2f¥n", x, asin(x));

   return 0;
}
```

● 参照→acos()、atan()、atan2()、cos()、sin()、tan()

10	式を評価して、その結果が偽である場合にプログラムを中止する **assert()**
ヘッダ	assert.h
書式	void assert (int *expr*);
引数	*expr* ……評価する式や実行する関数呼び出し。

● 解説

引数の式の値が偽（0）の場合に、標準出力にエラーメッセージを表示し、さらにabort()を呼び出してプログラムを終了させます。この関数は、主にデバッグのときに使い、通常はNDEBUGが定義されているときには評価されません。

● 注意

開発ツールによってはリリースモードのコンパイル時に自動的にNDEBUGが定義され、assert()が評価されないことがあります。

互換性

処理系によっては、assert()は、マクロとして実装されています。Windowsでは、assert()は、リリース/デバッグの両バージョンのC標準ライブラリで有効で、ほかに、_DEBUGフラグが定義されているときに式を評価するマクロ_ASSERTと_ASSERTEとが定義されています。

例

次の例は、assert()を使って値が0でないかどうか調べ、割り算の分母が0のときには演算を行わないようにします。

```
#include <stdio.h>
#include <assert.h>

int main(int argc, char *argv[])
{
    int n1, n2;
    printf("整数値を入力してください>");
    scanf("%d", &n1);
    printf("整数値を入力してください>");
    scanf("%d", &n2);
    printf("%d * %d = %d¥n", n1, n2, n1 * n2);
    printf("%d + %d の余り= %d¥n", n1, n2, n1 + n2);
    /* n2が0でないか調べる。assert(n2);でも良い */
    assert(n2 != 0);
    printf("%d / %d = %f¥n", n1, n2, (float)n1/(float)n2);
    printf("%d / %d の余り= %d¥n", n1, n2, (n1 % n2));
    return 0;
}
```

11 アークタンジェントを計算する atan()

ヘッダ	math.h
書式	double atan(double *x*);
引数	*x* ………アークタンジェントを求める角度（ラジアン単位）。
戻り値	アークタンジェントの値（ラジアン単位）。

解説

引数で指定した角度に対応するアークタンジェント（逆正接）を計算します。

例… 次の例は、入力された角度に対するアークタンジェントの値を表示します。

```
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ];
   double x;
   printf("角度（ラジアン、-1～1の範囲の実数）= ");
   scanf("%s", buff);
   x = atof(buff);
   printf("%.2fのアークタンジェントの値= %.2f¥n", x, atan(x));
   return 0;
}
```

参照→acos()、asin()、atan2()、cos()、sin()、tan()

12　アークタンジェントを計算する atan2()

ヘッダ	math.h
書式	double atan2(double *y*, double *x*);
引数	*x*、*y* ……アークタンジェントを求める角度をy/xで指定します。
戻り値	アークタンジェントの値（ラジアン単位）。

解説

2つの変数*y*と*x*で指定された角度に対するアークタンジェント（逆正接）を計算します。

例… 次の例は、入力された角度に対するアークタンジェントの値を表示します。

```
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv[])
{
   float x, y;
   printf("角度（y/xのxとy）= ");
   scanf("%f %f", &x, &y);
   printf("%1.2fのアークタンジェントの値= %.2f¥n", y/x, atan2(y, x));

   return 0;
}
```

参照→atan()

13	プログラムが正常終了した時に呼び出す関数を登録する atexit()
ヘッダ	stdlib.h
書式	int atexit(void (**func*)(void));
引数	*func* ……呼び出す関数名。
戻り値	関数登録が成功すると0。 失敗したときには-1（エラーコードをerrnoにセットする）

● 解説

プログラムが正常終了したときに呼び出される関数を登録します。プログラムの正常終了は、exit()が呼び出されたとき、またはプログラムがmain()からリターンして終了するときに発生します。登録した関数は、登録した順番とは逆の順番で呼び出されます。

● 注意

この関数を使って登録した関数に引数の値は渡されないので、登録する関数は引数なしの関数でなければなりません。

● 参照→exit

● エラーコード… ENOMEM

● 互換性

Windowsには同じ目的の独自の関数_onexit()があります。

● 例

次の例は、入力された数値に応じて終了時に実行する関数を変える例です。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void funcexit(void)
{
   printf("プログラムを正常終了しました。¥n");
}

void funcgood(void)
{
   printf("計算が正しく実行できました。¥n");
```

```
}

int main(int argc, char* argv[])
{
    char buff[BUFSIZ];
    double v;
    /* 終了後に実行する関数をあらかじめ登録する */
    atexit(funcexit);

    /* プログラムのメインロジック */
    printf("数値を入力してください>");
    scanf("%s", buff);
    v = atof(buff);

    /* nが0にきわめて近い数ならプログラムを終了する */
    if (fabs(v) < 0.001)
    {
        printf("%fでは計算できません¥n", v);
        exit (0);
    }

    /* 終了後に実行する最初の関数をあらかじめ登録する */
    atexit(funcgood);
    printf("365 / %f は=%f¥n", v , 365.0/v);

    return 0;
}
```

14 文字列をdouble型の数値に変換する atof()

ヘッダ	stdlib.h
書式	double atof(const char **nptr*);
引数	*nptr* ……実数に変換する数が含まれる文字列のポインタ。
戻り値	変換された実数値。変換できない場合は0.0。

● 解説

引数の文字列のはじめの数値部分をdouble型の実数に変換します。

文字列のはじめの数値部分とは、文字列の中の数値として解釈できない空白以外の文字の直前までです。文字列のはじめの部分に数値の文字がない場合には、この関数は0.0を返します。文字列のはじめの部分に数値の文字がないかどうか調べたいときには、strtod()を使うことができます。

例

次の例は、文字列を実数値に変換するプログラムの例です。

```
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ];
   double v;

   printf("数値を入力してください>");
   scanf("%s", buff);

   v = atof(buff);
   printf("365 / %f は=%f¥n", v , 365.0/v);

   return 0;
}
```

15 文字列をint型に変換する atoi()

ヘッダ	stdlib.h
書式	int atoi(const char **nptr*);
引数	*nptr* ……整数に変換する数が含まれる文字列のポインタ。
戻り値	変換された実数値。変換できない場合は0。

解説

引数の文字列のはじめの数値部分をint型の整数に変換します。文字列のはじめの部分に数値の文字がない場合にはこの関数は0を返します。文字列のはじめの部分に数値の文字がないかどうか調べたいときには、strtol()を使うことができます。

例

次の例は、ユーザーが入力した文字列を整数に変換するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ], *endptr;
   int v;

   printf("文字列を入力してください>");
   scanf("%s", buff);

   v = atoi(buff);
   printf("値は=%d¥n", v);

   v = (int)strtol(buff, &endptr, 0);

   if (endptr == &buff[0])
      printf("文字列%s は数値ではありません.¥n", buff);
   else
      printf("値は=%d¥n", v);

   return 0;
}
```

備考

int型の値の範囲は環境によって異なります。

参照→atof()

16	文字列をロング整数に変換する atol()
ヘッダ	stdlib.h
書式	long atol(const char **nptr*);
引数	*nptr* ……整数に変換する数が含まれる文字列のポインタ。
戻り値	変換された実数値。変換できない場合は0。

解説

引数の文字列のはじめの数値部分をlong int型の整数に変換します。

文字列のはじめの部分に数値の文字がない場合にはこの関数は0を返します。文字列のはじめの部分に数値の文字がないかどうか調べたいときには、strtol()を使うことができます。

● 例

次の例はユーザーが入力した文字列を数値に変換するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ], *endptr;
   long v;

   printf("文字列を入力してください>");
   scanf("%s", buff);

   v = atol(buff);
   printf("値は=%ld¥n", v);

   v = (int)strtol(buff, &endptr, 0);

   if (endptr == &buff[0])
      printf("文字列%s は数値ではありません.¥n", buff);
   else
      printf("値は=%ld¥n", v);

   return 0;
}
```

● 参照→atof()

17 ソケットに名前を付ける bind()

ヘッダ	sys/types.h、sys/socket.h
書式	int bind(int *sockfd*, struct sockaddr **addr*, int *addrlen*);
引数	*sockfd*……バインドしようとしているソケットのデスクリプタ。 *addr*　……sockaddr構造体のポインタ。 *addrlen* …*addr*のサイズ。通常はsizeof (addr)を指定します。
戻り値	成功すると0。失敗すると-1。

● 解説

指定されたソケット*sockfd*にアドレス*addr*を結び付けます。

●注意

ファイルシステム上にソケットが作成されたら、ソケットが必要なくなったときにunlink()を呼び出してソケットを削除する必要があります。

●エラーコード

EACCES、EBADF、EFAULT、EINVAL、ELOOP、ENAMETOOLONG、ENOENT、ENOMEM、ENOTDIR、EROFS

●準拠… BSD

18　ソートされた配列をバイナリサーチする bsearch()

ヘッダ	stdlib.h
書式	void *bsearch(const void **key*, const void **base*, size_t *nmemb*, size_t *size*, int (**compar*)(const void *, const void *));
引数	*key*…………検索するキー。 *base*………検索対象の配列の最初の要素を指すポインタ。 *nmemb*……検索対象の配列要素の数。 *size*…………配列の1要素のサイズ（バイト単位）。 *compar*……比較関数名。
戻り値	検索に成功したときは配列の要素のうち一致した要素のポインタ。 見つからなかったときはNULL。

●解説

配列をバイナリサーチ（二分木検索）します。配列の内容は比較関数*compar*で昇順にソートされていなければなりません。

●注意

検索キーと一致した要素が複数ある場合、どの要素のポインタが返されるかはわかりません。

●例

次の例は、ランダムな整数値の配列を作り、それをソートしたあとで、特定の値を探すプログラムの例です。

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define MAXARRAY 20

int numarray[MAXARRAY];

int compare(const void *arg1, const void *arg2)
{
   return (*(int *)arg1 - *(int *)arg2);
}

int main(int argc, char *argv[])
{
   int i, v;

   srand((unsigned)time(NULL));

   printf("配列の内容¥n");
   for (i=0; i<MAXARRAY; i++)
   {
      numarray[i] = (rand() * 50) / RAND_MAX;
      printf("%d¥t", numarray[i]);
      if (((i + 1) % 5) == 0)
         printf("¥n");
   }

   qsort((void *)numarray, (size_t)MAXARRAY, sizeof(int), compare);

   printf("ソートした配列の内容¥n");
   for (i=0; i<MAXARRAY; i++)
   {
      printf("%d¥t", numarray[i]);
      if (((i + 1) % 5) == 0)
         printf("¥n");
   }

   printf("検索する数>");
   scanf("%d", &v);
   if (bsearch(&v, numarray, MAXARRAY, sizeof(int), compare) != NULL)
      printf("%d は配列にあります.¥n", v);
   else
      printf("%d は配列にありません.¥n", v);

   return 0;
}
```

● 参照→qsort()

19	メモリを動的に割り当てる calloc()
ヘッダ	stdlib.h、malloc.h / calloc.h
書式	void *calloc(size_t *n*, size_t *size*);
引数	*n* ………メモリを割り当てる配列の要素数。 *size* ……メモリを割り当てる配列の1要素のサイズ（バイト単位）。
戻り値	メモリの割り当てに成功したときは割り当てられたメモリを指すポインタ、失敗したときはNULL。

● 解説

*size*バイトの要素*n*個からなる配列にメモリを割り当て、割り当てられたメモリに対するポインタを返します。確保するメモリブロックのサイズは*n*×*size*です。確保されたメモリブロックの内容は0にクリアされます。

● 例

次の例は、ユーザーが入力した数だけ要素を持つ配列を動的に作成します。

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>  /* あるいは<calloc.h> */
#include <time.h>

int compare(const void *arg1, const void *arg2)
{
   return (*(int *)arg1 - *(int *)arg2);
}

int main(int argc, char *argv[])
{
   int i, n, *nums;
   printf("発生させる乱数の数を入力してください>");
   scanf("%d", &n);
   nums = (long *)calloc(n, sizeof(int));
   if(nums == NULL)
      printf("メモリ領域を割り当てられませんでした。¥n");

   srand((unsigned)time(NULL));
   for (i=0; i<n; i++)
   {
      nums[i] = rand();
   }

   qsort((void *)nums, (size_t)n, sizeof(int), compare);
```

```
    printf("ソートした配列の内容¥n");
    for (i=0; i<n; i++)
    {
        printf("%d¥t", nums[i]);
        if (((i + 1) % 5) == 0)
        printf("¥n");
    }
    free(nums);

    return 0;
}
```

● 参照→malloc()

20	引数の値を繰り上げた数を返す ceil()
ヘッダ	math.h
書式	double ceil (double *x*);
引数	*x* …………繰り上げた値を求める値。
戻り値	繰り上げた値。

● **解説**

引数の値を繰り上げた数を返します。たとえば、引数の値が2.3の場合、3.0が返されます。

● **例**

次の例はユーザーが入力した数値を繰り上げた結果を出力するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char* argv[])
{
    char buff[BUFSIZ];
    double v;
    printf("数値を入力してください>");
    scanf("%s", buff);
    v = strtod(buff, NULL);
    printf("%fを繰り上げた値は%f¥n", v, ceil(v));

    return 0;
}
```

21 プロセスを終了する _cexit()

ヘッダ	process.h
書式	void _cexit(void);

●解説

atexit()と_onexit()で登録された終了手続きを完全に実行し、呼び出し側に戻ります。プロセスは終了しません。

●互換性… Windows

●参照→atexit()、exit()、_onexit()

22 プロセスを終了する _c_exit()

ヘッダー	process.h
書式	void _c_exit(void);

●解説

後処理をしないで終了手続きを実行し、呼び出し側に戻ります。プロセスは終了しません。

●互換性… Windows

●参照→exit()

23 カレントディレクトリを変更する chdir()

ヘッダ	unistd.h、direct.h、dir.h
書式	int chdir(const char **path*);
引数	*path* ……変更後の新しいパスを表す文字列定数。
戻り値	成功したときは**0**、失敗すると**-1**。

●解説

カレントディレクトリを、指定したディレクトリ*path*に変更します。

●エラーコード

EACCES、EFAULT、EIO、ELOOP、ENAMETOOLONG、ENOENT、ENOMEM、ENOTDIR

(返されるエラーコードは、ファイルシステムによって異なる可能性があります。)

例

次の例はディレクトリを変更するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>

#ifdef WIN32
#include <direct.h>
#else
#include <unistd.h>
#endif

int main(int argc, char* argv[])
{
   char path[BUFSIZ];
   printf("ディレクトリを入力してください>");
   scanf("%s", path);
#ifdef WIN32
   _chdir(path);
   system("dir");
#else
   chdir(path);
   system("pwd");
#endif

   return 0;
}
```

24　カレントディレクトリを変更する _chdir()

ヘッダ	direct.h
書式	int _chdir(const char **path*);
引数	*path* ……変更後の新しいパスを表す文字列定数。
戻り値	成功したときは0、失敗すると-1。

解説

カレントディレクトリを、指定したディレクトリ*path*に変更します。

準拠… ISO C++

参照→chdir()

25 ファイルのアクセスモードを変更する chmod()

ヘッダ	io.h、sys/stat.h、sys/types.h
書式	int chmod(const char **path*, int *amode*);
引数	*path*……アクセスモードを変更するファイル名。 *amode*…アクセス許可を以下のフラグの組み合わせ（解説参照）で指定します。
戻り値	成功したときは0。失敗すると-1。

● 解説

ファイルのアクセスモードを変更します。

*amode*には以下のアクセス許可フラグの組み合わせを指定します。

▼ 表 amodeの値

amodeの値	意味
S_IWRITE	書き込み許可。
S_IREAD	読み出し許可。
S_IREAD | S_IWRITE	読み出しと書き込み許可（書き込み許可には読み出し許可も含まれる）。
S_ISUID	実行時のユーザーIDの設定許可。
S_ISGID	実行時のグループIDの設定許可。
S_ISVTX	スティッキービット（終了時にプログラムを記憶領域から削除しない）。
S_IRUSR	所有者は読み出し可能。
S_IWUSR	所有者は書き込み可能。
S_IXUSR(S_IEXEC)	所有者は実行と検索可能。
S_IRGRP	グループは読み取り可能。
S_IWGRP	グループは書き込み可能。
S_IXGRP	グループによる実行/検索。
S_IROTH	他人（others）は読み出し可能。
S_IWOTH	他人は書き込み可能。
S_IXOTH	他人は実行/検索可能。

● エラーコード

EACCES、ENOENT

●例

次の例は、ファイルの情報を取得した後に、ファイルのモードを変更するプログラムの例です。

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#ifdef WIN32
#include <io.h>
#endif

int dispmod(char *fname)
{
#ifdef WIN32
   struct _stat buf;
#else
   struct stat buf;
#endif
   int fh;
#ifdef WIN32
   if((fh = _open(fname, _O_RDONLY)) == -1)
#else
   if((fh = open(fname, O_RDONLY)) == -1)
#endif
   {
      printf("ファイル%s を開けません.¥n", fname);
      return -1;
   }
#ifdef WIN32
   if (_fstat(fh, &buf) != 0)
#else
   if (fstat(fh, &buf) != 0)
#endif
      printf("ファイル%s の情報を得られませんでした.¥n", fname);
   else
   {
      printf("ファイルの大きさ: %ld¥n", buf.st_size);
      printf("変更された時刻: %s", ctime(&buf.st_atime));
      printf("ファイルモード: %x¥n", buf.st_mode);
   }
#ifdef WIN32
```

```
    _close(fh);
#else
    close(fh);
#endif
    return 0;
}

int main(int argc, char *argv[])
{
    char fname[BUFSIZ];
    printf("ファイル名>");
    scanf("%s", fname);
    printf("変更前のファイルの状態：¥n");
    if (dispmod(fname))
        exit (1);

    /* ファイルを読み出し専用にする。 */
#ifdef WIN32
    if(_chmod(fname, _S_IREAD) == -1)
#else
    if(chmod(fname, S_IREAD) == -1)
#endif
        puts("モードを変更できません.¥n");
    else
        puts("モードを変更しました.¥n");

    printf("変更後のファイルの状態：¥n");
    if (dispmod(fname))
        exit (1);

    /* 読み書きモードに戻す。 */
#ifdef WIN32
    if(_chmod(fname, _S_IWRITE) == -1)
#else
    if(chmod(fname, S_IWRITE | S_IREAD) == -1)
#endif
        puts("モードを変更できません.");
    else
        printf("読み書きモードに変更しました¥n");
        printf("変更前に戻したファイルの状態：¥n");

    if (dispmod(fname))
        exit (1);

    return 0;
}
```

26	ファイルのアクセスモードを変更する _chmod()
ヘッダ	io.h、sys/stat.h、sys/types.h
書式	int _chmod(const char **filename*, int *amode*);
引数	*filename* …アクセスモードを変更するファイル名。 *amode* ……アクセス許可を以下のフラグの組み合わせで指定します。
戻り値	成功したときは0。失敗すると-1。

● 解説

ファイルのアクセスモードを変更します。アクセスモードには、_S_IWRITE（読み書き許可）または_S_IREAD（読み出し許可）を指定します。

● 準拠… ISO C++

● 参照→chmod()

27	ファイルの所有者を変更する chown()
ヘッダ	sys/types.h、unistd.h
書式	int chown(const char **path*, uid__ *owner*, gid__ *group*);
引数	*path* ………所有者を変更したいファイル名。 *owner* ……新しい所有者。 *group* ……新しいグループ。
戻り値	成功したときは0。失敗すると-1。

● 解説

引数*path*で指定されたファイルの所有者を変更します。

通常、スーパーユーザーだけがファイルの所有者を変更できます。ファイルの所有者は、その所有者が属しているグループのいずれかに、ファイルのグループを変更できます。スーパーユーザーは、任意のグループに変更できます。そのため、この関数はスーパーユーザーだけが使える場合があります。また、chown()はシンボリックリンクを追跡しません。

● エラーコード

ファイルシステムによって返されるエラーコードは異なります。たとえば、ENOENT、ENOTDIR、EACCESなどが返されます。

● 互換性

この関数はUNIX系OS特有の機能に関連した、ファイルシステムに依存する関数です。

28 ファイルのステータスをクリアする clearerr()

ヘッダ	stdio.h
書式	void clearerr(FILE **fp*);
引数	*fp* ………FILE構造体を指すポインタ

●解説

ファイル*fp*のエラーインジケータとEOF（End Of File）インジケータをリセットします。

29 プロセッサ時間を返す clock()

ヘッダ	time.h
書式	clock_t clock(void);
戻り値	成功したときはプログラムの開始から経過したプロセッサ時間、 そうでなければ-1。

●解説

プロセッサ時間を返します。返される時間はclock_t型なので、秒単位の時間を得るにはclock()が返した値をCLK_TCKで割ります。

●互換性

プログラムの開始時点でclock()が返す値は標準では定められていません。そのため、プログラムの実行中の経過時間を測定したいときには、時間を測定するコードブロックの前後でclock()を実行して得られた値の差を使うべきです。

●例

この例では二重ループにかかる時間を測定するために、clock()を2回呼び出します。

最初のclock()の呼び出しの結果はシステムによって異なるので、このプログラムの実行結果はシステムによって異なります。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void sleep(clock_t wait);

int main(int argc, char *argv[])
{
```

```
    int i, j, n;
    clock_t start, finish;
    double duration, v;

    printf("繰り返し回数は>");
    scanf("%d", &n);
    start = clock();
    v = 1.01;
    for(i=0; i<n; i++)
    {
        if (((i+1) % 10) == 0)
        {
            fprintf(stdout, ".");
            fflush(stdout);
        }
        for(j=0; j<n; j++)
        {
            v *= 1.0001;
        }
    }
    printf("¥n");

    finish = clock();
    duration = (double)(finish) / CLOCKS_PER_SEC;
    printf("プログラム実行時間は約%2.1f 秒です。¥n", duration);
    duration = (double)(finish - start) / CLOCKS_PER_SEC;
    printf("計算にかかった時間は約%2.1f 秒です。¥n", duration);

    return 0;
}
```

30 ファイルデスクリプタを閉じる close()

ヘッダ	unistd.h
書式	int close(int *fd*);
引数	*fd* ………開いているファイルのデスクリプタ。
戻り値	成功したときは0、エラーが発生したときは-1。

●解説

指定したファイルデスクリプタを閉じます（クローズします）。閉じたデスクリプタは、どのファイルも参照していない状態になり、再使用できます。

注意

通常、close()の戻り値はチェックしません。しかし、ファイルシステムのパフォーマンスを向上させるために、データの書き込みが実際に終わっていなくても書き込み関数が成功したことを示す値を返す可能性があります。そのような場合、この関数の戻り値をチェックしないでファイルがクローズしたものとみなすと、データを失う可能性があります。

エラーコード… EBADF

例

chmod()の例参照。

31 ファイルデスクリプタを閉じる _close()

ヘッダ	io.h
書式	int _close(int *fd*);
引数	*fd* ………開いているファイルのデスクリプタ
戻り値	成功したときは0、エラーが発生したときは-1。

解説

指定したファイルデスクリプタを閉じます

準拠… ISO C++

参照→close()

32 ディレクトリを閉じる closedir()

ヘッダ	sys/types.h、dirent.h
書式	int closedir(DIR **dir*);
引数	*dir*………閉じるディレクトリのディレクトリストリームデスクリプタ（通常opendir()が返した値）。
戻り値	成功したときは0、失敗したときは-1。

解説

*dir*に連結しているディレクトリストリームを閉じます。

● エラーコード… EBADF

● 例

opendir()の例参照。

33	コサインを計算する cos()
ヘッダ	math.h
戻り値	成功したときはコサインの値、そうでない場合は不定。
書式	double cos(double *x*);
引数	*x* ………コサインの値を計算したい角度（ラジアン単位）

● 解説

指定した角度に対するコサイン（余弦）を計算します。角度はラジアンで指定します。

● 例

sin()の例参照。

34	ハイパーボリックコサインを計算する cosh()
ヘッダ	math.h
書式	double cosh(double *x*);
引数	*x* ………ハイパーボリックコサインの値を計算したい角度（ラジアン単位）
戻り値	成功したときはハイパーボリックコサインの値、そうでない場合は戻り値は不定。

● 解説

指定した角度に対するハイパーボリックコサイン（双曲線余弦）を計算します。

● 例

sin()の例参照。

35　ファイルかデバイスを作成する
creat()

ヘッダ	sys/types.h、sys/stat.h、fcntl.h
書式	int creat(const char **path*, mode_t *mode*);
引数	*path* ……オープン／作成するファイルまたはデバイスの名前。 *mode* …ファイル／デバイスをオープンまたは作成するときのモード（open()参照）。
戻り値	成功したときは新しいファイルデスクリプタ、エラーが発生したときは-1。

● 解説

ファイルまたはデバイスを作成します。ファイルが存在するときには、ファイルをオープンしてサイズを0にします。この挙動は、O_CREAT | O_WRONLY | O_TRUNCを指定してopen()を呼び出すのと同じです。

● 注意

スペシャルファイルはopen()を使ってオープンできますが、creat()を使ってスペシャルファイルを作成することはできません。スペシャルファイルを作るときにはmknod()を使います。

この関数の名前はcreat()です。create()ではありません。

● エラーコード

EACCES、EEXIST、EFAULT、EISDIR、ELOOP、EMFILE、ENAMETOOLONG、ENFILE、ENOENT、ENOMEM、ENOSPC、ENOTDIR、EROFS、ETXTBSY

● 例

次の例は、ファイルを作成してデータを指定した回数だけ書き込むプログラムの例です。

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>

#ifdef WIN32
#include <io.h>
#else
#include <fcntl.h>
#endif
#include <stdio.h>

int main(int argc, char *argv[])
```

```
{
   int fh, i, n;
   char fname[BUFSIZ], buff[BUFSIZ];
   printf("ファイル名>");
   scanf("%s", fname);
   printf("書き込むデータ>");
   scanf("%s", buff);
   printf("書き込む回数>");
   scanf("%d", &n);

#ifdef WIN32
   fh = _creat(fname, _S_IREAD | _S_IWRITE);
#else
   fh = creat(fname, S_IREAD | S_IWRITE);
#endif
   if(fh == -1)
   {
      puts("データファイルを作成できませんでした");
      exit (1);
   }

   for (i=0; i<n; i++)
   {
#ifdef WIN32
      if (_write(fh, buff, strlen(buff)) == -1)
#else
      if (write(fh, buff, strlen(buff)) == -1)
#endif
         puts("データを書き込めませんでした");
   }
#ifdef WIN32
   _close(fh);
#else
   close(fh);
#endif

   return 0;
}
```

36 ファイルかデバイスを作成する _creat()

ヘッダ	io.h
書式	int _creat(const char **path*, int *mode*);
引数	*path* ……オープンまたは作成するファイルまたはデバイスの名前。 *mode* …ファイルまたはデバイスをオープンまたは作成するときのモード（open()参照）。
戻り値	成功したときは新しいファイルデスクリプタ、エラーが発生したときは-1。

● 解説

ファイルまたはデバイスを作成します。

● 準拠… ISO C++

● 参照→creat()

37 時刻のバイナリデータをASCII文字列に変換する ctime()

ヘッダ	time.h
書式	char *ctime(const time_t **timep*);
引数	*timep* …文字列に変換したい時刻の情報が入っているtime_t構造体のポインタ。
戻り値	時刻を表す文字列のポインタ。

● 解説

time_t型のカレンダー時刻を次に示す形式の文字列に変換します。

"Wek Mon dd hh:mm:ss yyyyy¥n"

Wekは曜日の略称で、Sun、Mon、Tue、Wed、Thu、Fri、Satのいずれかです。

Monは月の略称で、Jan、Feb、Mar、Apr、May、Jun、Jul、Aug、Sep、Oct、Nov、Decのいずれかです。

ddは日付、hhは時間、mmは分、ssは秒、yyyyyは年を表します。

● 例

asctime()の例参照。

38	時刻の間隔の計算 difftime()
ヘッダ	time.h
書式	double difftime(time_t *time1*, time_t *time0*);
引数	*time1*………終了時刻が入っているtime_t型の変数。 *time0*………開始時刻が入っているtime_t型の変数。
戻り値	秒単位の経過時間を表す倍精度実数。

●解説

時刻*time0*から時刻*time1*までの経過時間を返します。

●例

次の例は二重ループにかかる時間を測定するために、time()を2回呼び出します。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[])
{
   int i, j, n;
   time_t start, finish;
   double duration, v;
   printf("繰り返し回数は>");
   scanf("%d", &n);
   time(&start);
   v = 1.01;
   for(i=0; i<n; i++)
   {
      if (((i+1) % 10) == 0)
      {
         fprintf(stdout, ".");
         fflush(stdout);
      }
      for(j=0; j<n; j++)
      {
         v *= 1.0001;
      }
   }
   printf("¥n");
   time(&finish);
   duration = difftime(finish, start);
   printf("計算にかかった時間は約%6.2f 秒です。¥n", duration);

   return 0;
}
```

39 整数の割り算を行って商と余りを計算する div()

ヘッダ	stdlib.h
書式	div_t div(int *numer*, int *denom*);
引数	*numer* ……割られる数。 *denom* ……割る数。
戻り値	商と余りが入っているdiv_t型の値。

● 解説

numer/denomの商と余りを計算し、結果をdiv_t構造体に返します。

商はdiv_t.quot、余りはdiv_t.remで得ることができます。div_tは次のように定義されています。

```
typedef struct _div_t {
   int quot;    /* 商 (quotient)  */
   int rem;     /* 余り (remainder)  */
} div_t;
```

● 例

次の例は、整数の割り算を行って余りを求めるプログラムの例です。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[])
{
   int x,y;
   div_t div_result;
   printf("割られる数>");
   scanf("%d", &x);
   printf("割る数>");
   scanf("%d", &y);
   div_result = div(x, y);
   printf("%d/%d の商= %d、余り= %d¥n", x, y, div_result.quot, div_result.rem);

   return 0;
}
```

40 ファイルデスクリプタを複製する dup()

ヘッダ	unistd.h
書式	int dup(int *fd*);
引数	*fd* ………複製したいファイルハンドル。
戻り値	関数が成功したときは新しいデスクリプタ、エラーの場合は-1。

● 解説

指定したファイルデスクリプタの複製を作ります。*fd*は現在オープンしているファイルハンドルでなければなりません。作成した新しいデスクリプタはもとのデスクリプタと同じものとして使うことができます。

● エラーコード

EBADF、EMFILE

● 互換性… POSIX

● 参照→_dup()、dup2()

41 ファイルデスクリプタを複製する _dup()

ヘッダ	io.h
書式	int _dup(int *fd*);
引数	*fd* ………複製したいファイルハンドル。
戻り値	関数が成功したときは新しいデスクリプタ、エラーの場合は-1。

● 解説

指定したファイルデスクリプタの複製を作ります。*fd*は現在オープンしているファイルハンドルでなければなりません。作成した新しいデスクリプタはもとのデスクリプタと同じものとして使うことができます。

● 準拠… ISO C++

● 例

次の例は、stdoutに_write()を使って出力するプログラムの例です。

```
#include <io.h>
#include <stdio.h>

int main( void )
{
   int orgfh;
   FILE *fp;
   char str[] = "stdoutに出力¥n";

   orgfh = _dup( 1 );   /* fd=1は"stdout"  */

   if( orgfh == -1 )
   {
      exit( 1 );
   }

   _write( orgfh, str, strlen(str));

   /* もとのstdoutに戻す */
   _dup2( orgfh, 1 );

   puts( str );
}
```

42 ファイルデスクリプタを複製する dup2()

ヘッダ	unistd.h
書式	int dup2(int *oldfd*, int *newfd*);
引数	*oldfd* …複製したいファイルハンドル。これは現在オープンしているファイルハンドルでなければなりません。 *newfd* …*oldfd*を関連付けるファイルハンドル。
戻り値	関数が成功したときは新しいデスクリプタ、エラーの場合は-1。

● 解説

*oldfd*の複製として*newfd*を作成します。必要ならば最初に*newfd*をクローズします。

● エラーコード… EBADF、EMFILE

● 互換性… POSIX

43	ファイルデスクリプタを複製する **_dup2()**
ヘッダ	io.h
書式	int _dup2(int *oldfd*, int *newfd*);
引数	*oldfd* …複製したいファイルハンドル。 *newfd* …*oldfd*を関連付けるファイルハンドル。
戻り値	関数が成功したときは新しいデスクリプタ、エラーの場合は-1。

解説

*oldfd*の複製として*newfd*を作成します。必要ならば最初に*newfd*をクローズします。*oldfd*は現在オープンしているファイルハンドルでなければなりません。

準拠… ISO C++

例

_dup()の例参照。

44	ファイルシステム記述ファイルを閉じる **endmntent()**
ヘッダ	stdio.h、mntent.h
書式	int endmntent(FILE **fp*);
引数	*fp* ………ファイルシステム記述ファイルのファイルポインタ （通常はsetmntent()が返した値）。
戻り値	常に1。

解説

ファイルシステム記述ファイルを閉じます。

仕様… BSD

例

`getmntent()`の例参照。

参照→fopen()、getmntent()

45	プログラムを実行する execl()
ヘッダ	process.h
書式	int execl(const char **path*, const char **arg*, ...);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *arg* ……プログラムの引数（任意の数だけ指定）。
戻り値	この関数がリターンしたときは戻り値は-1（エラーが発生している）

● 解説

指定された実行可能なプログラムを実行します。そのときに、現在のプロセスイメージを新しいプロセスイメージで置き換えます。引数リスト（arg, ...）の最後はNULLでなければなりません。

この関数は成功すると呼び出し側のプロセスに制御を戻しません（戻り値も返しません）。

この関数は実行するファイルを環境変数PATHに従って検索しません。そのため、通常、実行するファイルを完全なパスで指定しなければなりません。PATHを検索してファイルを実行させたいときには、execlp()を使ってください。

● 例

次の例は、外部のプログラムを実行するプログラムの例です。

```
#include <stdio.h>
#include <errno.h>

#ifdef WIN32
#include <process.h>
#else
#include <unistd.h>
#endif

int main(int argc, char *argv[])
{
#ifdef WIN32
   if (_execl("C:¥¥DOS¥¥tree", "tree", "/A", NULL) == -1)
#else
   if (execl("/bin/ls", "ls", "-l", NULL) == -1)
#endif
   {
      switch(errno) {
      case E2BIG:
         puts("引数リストと環境設定に必要な領域が32KBを超えています。");
         break;
```

```
        case EACCES:
            puts("ファイルにロック違反か共有違反が発生しています。");
            break;
        case EMFILE:
            puts("開いているファイルが多すぎます。");
            break;
        case ENOENT:
        puts("ファイル名またはパス名が見つかりません。");
            break;
        case ENOEXEC:
            puts("実行可能ファイルでないか実行可能ファイルの書式が無効。");
            break;
        case ENOMEM:
            puts("メモリ不足かメモリを割り当てられません。");
            break;
        default:
        puts("_execl()が失敗しました.");
        }
    }
    return 0;
}
```


46 プログラムを実行する _execl()

ヘッダ	process.h
書式	intptr_t _execl(const char **path*, const char **arg*, ...);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *arg* ……プログラムの引数（任意の数だけ指定）。
戻り値	この関数がリターンしたときは戻り値は-1（エラーが発生している）

● 解説

指定された実行可能なプログラムを実行します。

● 参照→_execl()

47	プログラムを実行する execle()
ヘッダ	process.h
書式	int execle(const char **path*, const char **arg* , ..., char * const *envp*[]);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *arg* ……プログラムの引数を任意の数だけ指定します。引数リストの最後はNULLでなければなりません。 *envp*……環境設定の配列のポインタ。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● 解説

指定されたプログラムを、指定された環境で実行します。そのため、通常、実行するファイルを完全なパスで指定しなければなりません。

● 例

次の例は、環境変数ENVSTR1とENVSTR2を設定したあとで、getenv()のサンプルプログラムを実行します。

```
#include <stdio.h>
#include <errno.h>
#ifdef WIN32
#include <process.h>
#else
#include <unistd.h>
#endif

char *envstr[] =
{
   "ENVSTR1=environment string 1",
   "ENVSTR2=envstr2",
   NULL
};

int main(int argc, char *argv[])
{
#ifdef WIN32
   if (_execle("H:¥¥getenv", "getenv", "ENVSTR1", "ENVSTR2", NULL, envstr)
      == -1)
#else
   if (execle("/usr/local/bin/getenv", "getenv", "ENVSTR1", "ENVSTR2",
      NULL, envstr) == -1)
```

```
#endif
    {
        switch(errno) {
        case E2BIG:
            puts("引数リストと環境設定に必要な領域が32KBを超えています。");
            break;
        case EACCES:
            puts("指定されたファイルにロック違反か共有違反が発生しています。");
            break;
        case EMFILE:
            puts("開いているファイルが多すぎます。");
            break;
        case ENOENT:
            puts("ファイル名またはパス名が見つかりません。");
            break;
        case ENOEXEC:
            puts("実行可能ファイルでないか、実行可能ファイルの書式が無効です。");
            break;
        case ENOMEM:
            puts("メモリ不足かメモリを割り当てられません。");
            break;
        default:
            puts("_execl()が失敗しました.");
        }
    }
    return 0;
}
```

● **参照**→execve()

プログラミング豆知識

コンピュータの時刻

コンピュータ内部の時計は、現地時間（ローカルタイム）にもUTC（世界協定時刻）にも設定することができます。コンピュータ内部の時計を現地時間に設定したときには、通常、OSがUTCとその地域の時間帯（タイムゾーン）とのオフセットを管理し、現地時間もUTCも利用できるようにします。

48 プログラムを実行する _execle()

ヘッダ	process.h
書式	int _execle(const char **path*, const char **arg* , ..., char * const *envp*[]);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *arg* ……プログラムの引数を任意の数だけ指定します。 引数リストの最後はNULLでなければなりません。 *envp*……環境設定の配列のポインタ。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● 解説

指定されたプログラムを、指定された環境で実行します。

● 参照→execle()

49 プログラムを実行する execlp()

ヘッダ	process.h
書式	int execlp(const char **file*, const char **arg*, ...);
引数	*file* ……実行するファイル名。 *arg* ……実行するファイルに渡す引数。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● 解説

指定されたファイルを実行します。指定されたファイル名がスラッシュ（/）を含んでいない場合、環境変数PATHで指定されたパスで実行可能なファイルを検索します。UNIX系OSでは、通常、環境変数PATHが設定されていない場合、デフォルトパスとして/bin:/usr/binが使われます。

● 例

次の例は、外部のプログラムを実行するプログラムの例です。

```
#include <stdio.h>
#include <errno.h>
#ifdef WIN32
#include <process.h>
#else
#include <unistd.h>
#endif

int main(int argc, char *argv[])
{
#ifdef WIN32
   if (_execlp("tree", "tree", "/A", NULL) == -1)
#else
   if (execlp("ls", "ls", "-l", NULL) == -1)
#endif
   {
      switch(errno) {
      case E2BIG:
         puts("引数リストと環境設定に必要な領域が32KBを超えています。");
         break;
      case EACCES:
         puts("指定されたファイルにロック違反か共有違反が発生しています。");
         break;
      case EMFILE:
         puts("開いているファイルが多すぎます。");
         break;
      case ENOENT:
         puts("ファイル名またはパス名が見つかりません。");
         break;
      case ENOEXEC:
         puts("実行可能ファイルでないか、実行可能ファイルの書式が無効です。");
         break;
      case ENOMEM:
         puts("メモリ不足かメモリを割り当てられません。");
         break;
      default:
         puts("_execl()が失敗しました。");
      }
   }
   return 0;
}
```

● 参照→execve()

50	プログラムを実行する _execlp()
ヘッダ	process.h
書式	int _execlp(const char **file*, const char **arg*, ...);
引数	*file* ……実行するファイル名。 *arg* ……実行するファイルに渡す引数。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● 解説

指定されたファイルを実行します。

● 参照→execlp()

51	プログラムを実行する execv()
ヘッダ	process.h
書式	int execv(const char **path*, char *const *argv*[]);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *argv* ……プログラムの引数の配列のポインタ(配列の最後の要素はNULL)。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● 解説

指定されたプログラムを、配列で指定された引数を伴って実行します。

● 例

次の例は、外部のプログラムを実行するプログラムの例です。

```
#include <stdio.h>
#include <errno.h>
#ifdef WIN32
#include <process.h>
#include <string.h>
#else
#include <unistd.h>
#endif

char *args[] =
{
   "tree",
```

```
    "-1",
    NULL
};

int main(int argc, char *argv[])
{
#ifdef WIN32
    strcpy(args[1], "/A");
    if (_execv("C:¥¥DOS¥¥tree", args) == -1)
#else
    if (execv("/usr/bin/tree", args) == -1)
#endif
{
    switch(errno) {
        case E2BIG:
            puts("引数リストと環境設定に必要な領域が32KBを超えています。");
            break;
        case EACCES:
            puts("指定されたファイルにロック違反か共有違反が発生しています。");
            break;
        case EMFILE:
            puts("開いているファイルが多すぎます。");
            break;
        case ENOENT:
            puts("ファイル名またはパス名が見つかりません。");
            break;
        case ENOEXEC:
            puts("実行可能ファイルでないか、実行可能ファイルの書式が無効です。");
            break;
        case ENOMEM:
            puts("メモリ不足かメモリを割り当てられません。");
            break;
        default:
            puts("_execl()が失敗しました.");
        }
    }
    return 0;
}
```

● 参照→execve()

52 プログラムを実行する _execv()

ヘッダ	process.h
書式	int _execv(const char **path*, char *const *argv*[]);
引数	*path*……実行するプログラムを示す文字列のポインタ。 *argv*……プログラムの引数の配列のポインタ(配列の最後の要素はNULL)。
戻り値	この関数がリターンしたときは-1（エラーが発生している)。

● 解説

指定されたプログラムを、配列で指定された引数を伴って実行します。

● 参照→execv()

53 プログラムを実行する execve()

ヘッダ	process.h
書式	int execve(const char **cmdname*, const char **argv*, const char **envp*);
引数	*cmdname*…実行するプログラムを示す文字列のポインタ。 *argv*………プログラムの引数の配列のポインタ（配列の最後の要素はNULL)。 *envp*………環境設定の配列のポインタ（配列の最後の要素はNULL)。
戻り値	この関数がリターンしたときは-1（エラーが発生している)。

● 解説

指定されたプログラムを、指定された環境で実行します。そのため、通常、実行するファイルを完全なパスで指定しなければなりません。

● 例

execv()とexecle()の例参照。

54 プログラムを実行する _execve()

ヘッダ	process.h
書式	int _execve(const char **cmdname*, const char **argv*, const char **envp*);
引数	*cmdname* …実行するプログラムを示す文字列のポインタ。 *argv*…………プログラムの引数の配列のポインタ（配列の最後の要素はNULL）。 *envp* ………環境設定の配列のポインタ（配列の最後の要素はNULL）。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● 解説

指定されたプログラムを、指定された環境で実行します。

● 参照→execve()

55 プログラムを実行する execvp()

ヘッダ	process.h
書式	int execvp(const char **path*, char *const *argv*[]);
引数	*path*……実行するプログラムを示す文字列のポインタ。 *argv* ……プログラムの引数の配列のポインタ(配列の最後の要素はNULL)。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● 解説

指定されたプログラムを指定された環境で実行します。指定されたファイル名がスラッシュ（/）を含んでいない場合、環境変数PATHで指定されたパスで実行可能なファイルを検索します。UNIX系OSでは、通常、環境変数PATHが設定されていない場合、デフォルトパスとして/bin:/usr/binが使われます。

● 例

execv()とexeclp()の例参照。

56	プログラムを実行する _execvp()
ヘッダ	process.h
書式	int _execvp(const char **path*, char *const *argv*[]);
引数	*path*……実行するプログラムを示す文字列のポインタ。 *argv*……プログラムの引数の配列のポインタ(配列の最後の要素はNULL)。
戻り値	この関数がリターンしたときは-1(エラーが発生している)。

解説

指定されたプログラムを指定された環境で実行します。

参照→execvp()

57	プロセスを終了する exit()
ヘッダ	process.h / stdlib.h
書式	void exit(int *status*);
引数	*status* …終了状態を示す整数値。通常、プロセスが正常終了する時は0、エラーのときはそれ以外の値。

解説

呼び出し側プロセスを終了します。この関数は、atexit()と_onexit()で登録された関数があれば、それを呼び出します。そして、すべてのファイルのバッファをフラッシュします(これらの処理を後処理と呼びます)。それから、この関数はプロセスを終了します。

*status*に指定できる定数として、EXIT_FAILUREとEXIT_SUCCESSが定義されています。

後処理をしないでプロセスを終了する関数として、_exit()が用意されています。

互換性

Windowsにはプロセスの終了に関連する関数として、_cexit()と_c_exit()があります。

例

atexit()の例参照。

参照→EXIT_FAILURE、EXIT_SUCCESS

58 現在のプロセスを終了させる _exit()

ヘッダ	unistd.h
書式	void _exit(int *status*);
引数	*status* …プロセスの終了状態として親プロセスに対して返す値。

●解説

プロセスを直ちに終了させます。プロセスが所有していてオープンしているデスクリプタはすべて閉じられ、プロセスが所有する子プロセスはすべてプロセス番号1(initプロセス)が継承し、この関数を呼び出したプロセスの親プロセスに対してシグナルSIGCHLDが送出されます。

●注意

_exit()はプロセスを直ちに終了させます。終了するときに、atexit()を使ってあらかじめ登録した関数を呼び出したり、I/Oバッファをフラッシュしてから終了したいときには、exit()を使ってください。

59 現在のプロセスを終了させる _Exit()

ヘッダ	stdlib.h
書式	void _Exit(int *status*);
引数	*status* …プロセスの終了状態として親プロセスに対して返す値。

●解説

プロセスを直ちに終了させます。プロセスが所有していてオープンしているデスクリプタはすべて閉じられ、プロセスが所有する子プロセスはすべてプロセス番号1(initプロセス)が継承し、この関数を呼び出したプロセスの親プロセスに対してシグナルSIGCHLDが送出されます。

●準拠… C99

プログラミング豆知識

プログラムの戻り値

C/C++では、プログラムが終了したときにOSに返されるプログラムの戻り値は、main()が返した値です。プログラムが正常終了したときにはreturn 0;を使って関数main()がゼロを返し、そうでないときはゼロ以外の値を返すようにすると、OSのシェルスクリプトなどでプログラムが終了したときの結果に応じてその後の操作を変更することができます。

60	指数を計算する exp()
ヘッダ	math.h
書式	double exp(double *x*);
引数	*x* ………指数値を求める値。

● 解説

指定された値に対する指数値を返します。

● 例

次の例は、入力された値に対する指数を計算するプログラムの例です。

```
#include <math.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   double y;
   float x;

   printf("値>");
   scanf("%f", &x);

   y = exp((double)x);
   printf("exp(%f) = %f¥n", x, y);

   return 0;
}
```

● **参照**→sqrt()

61	浮動小数点実数の絶対値を計算する fabs()
ヘッダ	math.h
書式	double fabs(double *x*);
引数	*x* ………浮動小数点実数の絶対値を求める値。

● **解説**… fabs()関数は浮動小数点実数*x*の絶対値を返します。

● **参照**→abs()、ceil()、floor()、labs()

62 ファイルのモードを変更する fchmod()

ヘッダ	sys/types.h、sys/stat.h
書式	int fchmod(int *fd*, mode_t *mode*);
引数	*fd* ………モードを変更するファイルディスクリプタ。 *mode* …ファイルにセットするモードフラグ（解説参照）。
戻り値	成功したときは0、失敗すると-1。

●解説

指定されたファイルのモードを変更します。ファイルに設定できるモードはファイルシステムに依存します。

*mode*には、次の表のフラグをorで組み合わせて指定します。

▼表 modeのフラグ

modeのフラグ	値	意味
S_ISUID	04000	実行時のセットユーザーID(set user ID)。
S_ISGID	02000	実行時のセットグループID(set group ID)。
S_ISVTX	01000	スティッキー(sticky)ビット。
S_IRUSR(S_IREAD)	00400	所有者による読み取り。
S_IWUSR(S_IWRITE)	00200	所有者による書き込み。
S_IXUSR(S_IEXEC)	00100	所有者による実行/検索。
S_IRGRP	00040	グループによる読み取り。
S_IWGRP	00020	グループによる書き込み。
S_IXGRP	00010	グループによる実行/検索。
S_IROTH	00004	他人(others)による読み取り。
S_IWOTH	00002	他人による書き込み。
S_IXOTH	00001	他人による実行/検索。

●エラーコード

エラーコードはファイルシステムによって異なります。主なエラーは次のとおりです。

EBADF、EIO、EPERM、EROFS

●準拠

POSIX（_POSIX_MAPPED_FILESか_POSIX_SHARED_MEMORY_OBJECTSのいずれかが定義されている必要があります。）

●参照→chmod()、_chmod()、_wchmod()

63	ファイルを閉じる **fclose()**
ヘッダ	stdio.h
書式	int fclose(FILE **fp*);
引数	*fp* ………閉じるファイルストリーム（通常はfopen()が返した値）。
戻り値	正常に終了したときは0、そうでなければEOF。

● 解説

指定されたファイルを閉じます。出力ストリームの場合は、ストリームが閉じられる前にバッファのデータが書き込まれます。ただし、ハードディスクなどの物理的なメディアに実際に書き込まれるタイミングはシステムに依存します。

● エラーコード… EBADF

● 参照→close()、fflush()、fopen()、setbuf()

64	ファイルポインタの位置がEOFであるかどうか調べる **feof()**
ヘッダ	stdio.h
書式	int feof(FILE **fp*);
引数	*fp* ………ファイルのデスクリプタ。
戻り値	EOFであれば0以外の値、そうでなければ0。

● 解説

指定されたストリームの現在のファイルポインタの位置がファイルの最後（EOF）であるかどうか調べます。

● 例

次の例は、カレントディレクトリにあるファイルtest.txtを1バイトずつ読み込んでは標準出力に出力するという作業をファイルの最後まで繰り返す例です。

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    FILE *stream;
    int c;
    stream = fopen("./test.txt", "r");
```

```
    if (stream == NULL) {
        fprintf(stdout, "ファイルを開くことができません.¥n");
        return;
    }
    while (!feof(stream))
    {
        c = fgetc(stream);
        fputc(c, stdout);
    }
    fclose(stream);

    return 0;
    }
```

65 ファイルのエラーをテストする ferror()

ヘッダ	stdio.h
書式	int ferror(FILE **fp*);
引数	*fp* ………テストするファイル（普通は、fopen()が返した値）。
戻り値	エラー指示子がセットされていれば0以外の値、そうでなければ0。

● 解説

*fp*で示されるファイルのエラー指示子をテストします。エラー指示子は、関数clearerr()を使ってリセットすることができます。

● 例

次の例は、すでに開いている入力ストリームistreamと出力ストリームostreamのエラーを調べて、エラーがあればストリームからの読み書きのループを終了する例です。

```
while (!feof(istream))
{
    if (ferror(istream))
        break;
    c = fgetc(stream);
    fputc(c, ostream);
    if (ferror(ostream))
        break;
}
```

● 参照→open()

66 ファイルをフラッシュする fflush()

ヘッダ	stdio.h
書式	fflush(FILE **fp*);
引数	*fp* ………フラッシュするファイル（普通は、fopen()が返した値）。
戻り値	成功したときは0、そうでなければEOF。

● 解説

バッファリングされたデータをすべて強制的に書き込んで更新します。*fp*にNULLを指定すると、開いているすべての出力ストリームをフラッシュします。ハードディスクなど物理的なメディアへの実際の書き込みのタイミングは、システムに依存します。

● エラーコード… EBADF

● 例

次の例は、すでに開いている出力ストリームostreamにデータを1バイト書きこみます。

```
fputc(c, ostream);
fflush(ostream);
```

● 参照→fclose()、fopen()、setbuf()、write()

67 ファイルから文字を読み出す fgetc()

ヘッダ	stdio.h
書式	int fgetc(FILE **fp*);
引数	*fp* ………文字を読み出すファイル（普通は、fopen()が返した値）。
戻り値	読み出した文字。エラーかファイルの終わりに達した場合はEOF。

● 解説

*fp*から次の文字を読み出します。文字は、unsigned charとして読み込んで、intにキャストして返します。

● 例

次の例は、カレントディレクトリにあるファイルtest.txtを1バイトずつ読み込んでは標準出力に出力するという作業をファイルの最後まで繰り返す例です。

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   FILE *stream;
   int c;
   stream = fopen("./test.txt", "r");
   if (stream == NULL) {
      fprintf(stdout, "ファイルを開くことができません.¥n");
      return;
   }
   while (!feof(stream))
   {
      c = fgetc(stream);
      fputc(c, stdout);
   }
   fclose(stream);

   return 0;
}
```

● **参照**→ferror()、fopen()、fread()、fseek()、puts()、read()、scanf()、write()

68 ファイルポインタの現在の位置を取得する fgetpos()

ヘッダ	stdio.h
書式	int fgetpos(FILE **fp*, fpos_t **pos*);
引数	*fp* ………位置を取得するストリーム。 *pos* ……位置インジケータを保存するfpos_t構造体のポインタ。
戻り値	成功したときは0。そうでないときは-1。

● **解説**

ファイルポインタの現在の位置（ファイルの先頭からのオフセット）の値を、引数*pos*が指すfpos_t構造体に保存します。

● **エラーコード**… EBADF、EINVAL

● **例**

次の例では、fsetpos()でファイルポインタの位置を10バイトの位置にセットしたあと、データを5バイト読み込んでから現在のファイルポインタの位置を取得します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   FILE *stream;
   int c, i;
   fpos_t pos;

   stream = fopen("./test.txt", "r");
   if (stream == NULL) {
      fprintf(stdout, "ファイルを開くことができません.¥n");
      return;
   }

   /* ファイルポインタを10バイトの位置にセットする */
   pos = 10;
   fsetpos(stream, &pos);
   /* 5バイト読み出す */
   for (i=0; i<5; i++)
   {
      c = fgetc(stream);
   }

   /* 現在のファイルポインタの位置を取得する */
   fgetpos(stream, &pos);
   fclose(stream);
   printf("ファイルポインタの位置=%ld", pos);

   return 0;
}
```

プログラミング豆知識

バイナリデータ

バイナリデータは日付と時刻やグラフィックスイメージのような比較的複雑な値を、システムの内部で効率的に使えるように表現した値で、通常は人間がその値を目で見てもその意味がすぐには明白にならないような値です。バイナリデータをあえてそのまま表示するときには、一般的には16進数で表示します。

69	ファイルから文字列を読み出す fgets()
ヘッダ	stdio.h
書式	char *fgets(char **s*, int *size*, FILE **fp*);
引数	*s* ………読み込んだ文字列を保存するための文字列バッファのポインタ。 *size* ……読み出す最大の長さ（バッファsの長さ以下の値）。 *fp* ………読み出すファイルのポインタ。
戻り値	成功したときは読み込んだ文字列のポインタ、エラーが発生するかファイルの終端に達したときはNULL。

● **解説**

最大で*size*-1文字の文字を*fp*から読み出し、バッファ*s*に保存します。EOFまたは改行文字を読み出すと読み出しを中止します。読み込まれた改行文字はそのままバッファに保存されバッファの最後に¥0が追加されます。

gets()は改行文字をNULL文字に置き換えます。

エラーが発生したかどうかを調べるには、feof()またはferror()を使います。

● **例**

次の例では、名前を文字列として読み出します。

```
#include <stdio.h>
#include <locale.h>

int main(int argc, char *argv[])
{
   char urname[256];
   printf("名前は？>");
   fgets(urname, 256, stdin);

   /* 最後の改行を削除する */
   urname[strlen(urname)-1] = '¥0';

   printf("Hello %s! 元気でやってるかい？¥n", urname);

   return 0;
}
```

70 ファイルのデスクリプタを返す
fileno()

ヘッダ	stdio.h
書式	int fileno(FILE **fp*);
引数	*fp* ………デスクリプタを調べるファイルのポインタ。

● 解説

引数*stream*を調べ、そのデスクリプタである整数値を返します。

● 例

次の例は開いたファイルのファイル番号を標準出力に出力する例です。

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    FILE *filep;
    filep = fopen("./test.txt", "w");
    if (filep == NULL)
    {
        fprintf(stderr, "ファイル/etc/fstabを開けません.¥n");
        return -1;
    }
    printf("fileno=%d¥n", fileno(filep));
    fclose(filep);
    return 0;
}
```

● 参照→open()

71 ファイルのデスクリプタを返す
_fileno()

ヘッダ	stdio.h
書式	int _fileno(FILE **fp*);
引数	*fp* ………デスクリプタを調べるファイルのポインタ

● 解説

引数*fp*を調べ、そのデスクリプタである整数値を返します。

● 準拠… ISO C++

● 参照→fileno()

72 引数の値を越えない最大の整数値を返す floor()

ヘッダ	math.h
書式	double floor(double *x*);
引数	*x* ………この値を超えない最大の整数値を求める値。
戻り値	引数の値を越えない最大の整数（倍精度実数）。

●解説

引数の値を越えない最大の整数を求めて、倍精度実数として返します。

●例

次の例は、入力された実数値を超えない最大の整数値を出力するプログラムの例です。

```
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[])
{
   float x;

   printf("実数値=");
   scanf("%f", &x);

   printf("¥n%fを超えない最大の整数値=%f¥n", x, floor(x));

   return 0;
}
```

●**参照**→abs()、ceil()、fabs()

プログラミング豆知識

名前空間

名前空間は大きなプロジェクトやプログラムコードの再利用のときに便利です。そのため、名前空間の概念は、XMLやJavaのようなほかの言語でも使われています。しかし、まだすべてのC++コンパイラが名前空間をサポートしているわけではありません。また、同じ「名前空間」と呼ばれていても、プログラミング言語間でその実態は異なります。

73	浮動小数点数の余りを計算する fmod()
ヘッダ	math.h
書式	double fmod(double *x*, double *y*);
引数	*x* ………割られる値。 *y* ………割る値。
戻り値	成功すれば余りの値。

● 解説

*x*を*y*で割った余りを計算します。

● エラーコード… EDOM

● 例

次の例は実数の割り算を行うプログラムの例です。

```
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[])
{
   float x, y;

   printf("分子=");
   scanf("%f", &x);

   printf("分母=");
   scanf("%f", &y);

   printf("%f / % f の余り=%f", x, y, fmod((double)x, (double)y));

   return 0;
}
```

74	ファイルを開く fopen()
ヘッダ	stdio.h
書式	FILE *fopen(char **path*, char **mode*);
引数	*path* ……オープンするファイルを示す文字列のポインタ。 *mode* …ファイルをオープンするときのモード（解説参照）。
戻り値	成功した時にはファイルポインタ。そうでない場合はNULL。

● 解説

指定されたファイルを指定されたモードで開きます。書き込みのためにファイルを開く場合、ファイルが存在していない場合にはファイルを新たに作成します。

*mode*には以下の値のいずれかを指定します。

▼ 表 modeの値

modeの値	意味
r	ファイルを読み出すために開く。
r+	読み出しと書き込みのために開く。
w	書き込みのためにファイルを開き、既存のファイルであればもとのファイルを長さ0にする。
w+	読み出しと書き込みのために開く。
a	書き込みのために開く。
a+	読み出しと書き込みのために開く。

また、*mode*にはbを指定することもでき、その場合は開くファイルがバイナリファイルであることを意味しますが、ANSI Cとの互換性のためだけに指定可能になっていて実際にはなんら影響を与えないことがあります。

● 互換性

プラットフォームや処理系によっては*mode*にさらにc、n、tを指定できる場合があります。

● エラーコード… EINVAL

● 例

fgetc()の例参照。

● 参照→fclose()、open()

75 書式を指定してデータをファイルに書き込む fprintf()

ヘッダ	stdio.h
書式	int fprintf(FILE **fp*, const char **fmt*, ...);
引数	*fp* ………出力するファイル。 *fmt* ……出力するときの書式。printf()参照。 ... ………出力する値のリスト。
戻り値	書き込んだ文字数。

● 解説

書式指定*fmt*に従ってファイルに出力します（printf()の解説参照）。

● 例

printf()の例参照。

● 参照→printf()、scanf()

76 文字をファイルに書き込む fputc()

ヘッダ	stdio.h
書式	int fputc(int *c*, FILE **fp*);
引数	*c* ………出力する文字。 *fp* ………出力するファイル。
戻り値	書き込まれた文字。エラーが発生したときはEOF。

● 解説

キャラクタ*c*を`unsigned char`にキャストして*fp*に書き込みます。書き込まれた文字は`int`にキャストして返されます。

● 例

feof()の例参照。

77 文字列をファイルに書き込む fputs()

ヘッダ	stdio.h
書式	int fputs(const char **s*, FILE **fp*);
引数	*s* ………出力する文字列。 *fp* ………出力するファイル。
戻り値	成功したときは負ではない数、エラーが発生したときはEOF。

● 解説

文字列*s*を*fp*に書き込みます。文字列の最後の¥0は出力しません。

● 例

次の例では、標準入力stdin（デフォルトではキーボード）から文字列を入力して、ファイルtest.datに保存します。入力文字列が"bye"であると、ファイルを閉じてプログラムが終了します。

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
   FILE *stream;
   char buff[200], buf[20];
   stream = fopen("./test.dat", "w");
   if (stream == NULL) {
      fprintf(stdout, "ファイルを開くことができません.¥n");
      return;
   }
   while (1)
   {
      printf("データ（byeで終了）>");
      if (fgets(buff, 128, stdin) == NULL)
         break;
      strncpy(buf, buff, 3);
      buf[3] = 0;
      if (strcmp(buf, "bye") == 0)
         break;
      fputs(buff, stream);
   }
   fclose(stream);

   return 0;
}
```

78 書式なしデータをファイルから読み出す fread()

ヘッダ	stdio.h
書式	size_t fread(void **ptr*, size_t *size*, size_t *n*, FILE **fp*);
引数	*ptr* ……データを保存するバッファのポインタ。 *size* ……データ1つのサイズ（バイト単位）。 *n* ………読み出す最大データの最大数。 *fp* ………読み出すファイル。
戻り値	読み込んだデータの数。エラーが発生した場合やファイルの最後に達した場合は0または*n*より小さい値。

解説

ファイル*fp*から、長さ*size*バイトのデータを最大*n*個読み出し、*ptr*で指定されたところに保存します。

ファイルの最後に達したか、エラーが発生したかを識別するには、`feof()`と`ferror()`を使います。

例

fwrite()の例参照。

79 メモリブロックを解放する free()

ヘッダ	stdlib.h、malloc.h
書式	void free(void **pmem*);
引数	*pmem* …解放するメモリブロック

解説

`calloc()`、`malloc()`、`realloc()`などを呼び出して割り当てたメモリブロックを解放します。

例

malloc()の例参照。

80 ファイルを開きなおす freopen()

ヘッダ	stdio.h
書式	FILE *freopen(char **path*, char **mode*, FILE **fp*);
引数	*path*……オープンするファイルを示す文字列のポインタ。 *mode* …ファイルをオープンするときのモード。fopen()参照。 *fp* ………開きなおすファイル。
戻り値	成功した時にはファイルポインタ、そうでない場合はNULL。

● 解説

ファイル*path*を開きファイル*fp*にそのファイルを接続します。もとのストリームが存在する場合には閉じられます。*mode*はfopen()と同じです。標準入出力（stderr、stdin、stdout）の対象を変更するときなどに使います。

● エラーコード… EINVAL

● 参照→fclose()、open()

81 浮動小数点実数を仮数と指数に分ける frexp()

ヘッダ	math.h
書式	double frexp(double *x*, int **expr*);
引数	*x* ………浮動小数点実数。 *expr*……指数を保存する変数のポインタ。
戻り値	仮数

● 解説

*x*を仮数（0.5以上1.0未満）と指数に分解します。指数を*expr*に入れ、仮数を返します。仮数部をm、指数部をnとしたとき、$x=m*2^n$になるようなmとnが返されます。

プログラミング豆知識

ファイルポインタ

プログラミング用語の「ポインタ」は、何かを指しているものを表します。単にポインタといえば、普通は変数やオブジェクトを指している値のことを指します。

「ファイルポインタ」は、ファイルの現在のアクセス位置を指しているものです。これは通常はファイルの先頭からの位置を示すバイト単位の値で、ファイルへのランダムなアクセスによく使われます。

82　書式付きデータをファイルから読み出す fscanf()

ヘッダ	stdio.h
書式	int fscanf(FILE **fp*, const char **fmt*, ...);
引数	*fp* ………開いているファイルストリームのポインタ。 *fmt* ……読み出すデータの書式。scanf()参照。 ... ………読み込んだデータを保存するための変数のポインタ。
戻り値	入力されたデータの個数。最初の読み出しの前にEOFがあって入力に失敗したときにはEOF。

● 解説

ファイル*fp*から書式*fmt*に従ってデータを読み出します（scanf()参照）。

● 例

scanf()の例参照。

83　ファイルポインタの位置を移動する fseek()

ヘッダ	stdio.h
書式	int fseek(FILE **fp*, long *offset*, int *origin*);
引数	*fp* ………開いているファイル。 *offset* …移動する量。originからのオフセット（バイト単位）。 *origin* …初期位置（解説参照）。
戻り値	成功したときは0、失敗したときは-1。

● 解説

ファイル*fp*の現在のファイルの位置を示すポインタを、*origin*で指定された位置から*offset*バイトの位置に移動します。

*origin*には次のいずれかの値を指定します。

▼ 表　originの値

originの値	意味
SEEK_CUR	ファイルポインタの位置。
SEEK_END	ファイルの終端。
SEEK_SET	ファイルの先頭。

●注意

初期位置*origin*にSEEK_CURやSEEK_ENDを指定して、それより前の位置に移動したいときは、*offset*に負の数を指定します。

●エラーコード… EBADF、EINVAL

●例

次の例では、ファイルを開いてファイルポインタをファイルの最後に移動した後、ファイルの最後から10バイトの位置に移動して1バイト取得し、それぞれの操作の前後で現在のファイルポインタの位置を調べて出力します。

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   FILE *stream;
   long pos;
   int c;
   stream = fopen("./test.dat", "r");
   if (stream == NULL) {
      fprintf(stdout, "ファイルを開くことができません.¥n");
      return;
   }

   /* ファイルポインタの位置を調べる */
   pos = ftell(stream);
   printf("%ld¥n", pos);

   /* ファイルの最後に移動する */
   if (fseek(stream, 0, SEEK_END))
      printf("シークできません.¥n");

   /* ファイルポインタの位置を調べる */
   pos = ftell(stream);
   printf("%ld¥n", pos);

   /* ファイルの最後から10バイトの位置に移動する */
   if (fseek(stream, 10, SEEK_SET))
      printf("シークできません.¥n");

   /* ファイルポインタの位置を調べる */
```

```
    pos = ftell(stream);
    printf("%ld¥n", pos);

    /* 1バイト取得する */
    c = fgetc(stream);
    printf("c=%c(%02x)¥n", (char)c, (char)c);

    /* ファイルポインタの位置を調べる */
    pos = ftell(stream);
    printf("%ld¥n", pos);
    fclose(stream);

    return 0;
}
```

84 ファイルポインタの位置を設定する fsetpos()

ヘッダ	stdio.h
書式	int fsetpos(FILE **fp*, fpos_t **pos*);
引数	*fp* ………開いているファイル。 *pos* ……ファイルポインタを移動する位置（バイト単位）。
戻り値	成功したときは0、失敗したときは-1。

● 解説

ファイルポインタの位置を、ファイルの先頭から*pos*バイトの位置に設定します。

● エラーコード… EBADF、EINVAL

● 例

fgetpos()の例参照。

● 参照→lseek()

85 ファイルの状態を得る fstat()

ヘッダ	sys/stat.h、unistd.h
書式	int fstat(int *fd*, struct stat **buf*);
引数	*fd* ………ファイルデスクリプタ（普通はopen()が返した値）。 *buf* ……ファイルの状態を示す情報を保存するstat構造体変数のポインタ。
戻り値	ファイルに関する情報の検索に成功したときは0、エラーの場合は-1。

● 解説

指定されたファイルに関する情報を取得します。stat構造体の例を以下に示します。

```
struct stat
{
   dev_t st_dev; /* デバイス */
   ino_t st_ino; /* inode */
   mode_t st_mode; /* アクセス権（モード） */
   nlink_t st_nlink; /* ハードリンクの数 */
   uid_t st_uid; /* 所有者のユーザーID */
   gid_t st_gid; /* 所有者のグループID */
   dev_t st_rdev; /* inodeデバイスの場合のデバイスの種類 */
   off_t st_size; /* 総サイズ（バイト単位） */
   unsigned long st_blksize; /* I/Oのためのブロックサイズ */
   unsigned long st_blocks; /* 確保されているブロックのサイズ */
   time_t st_atime; /* 最後にアクセスしたときの時刻 */
   time_t st_mtime; /* 最後に更新したときの時刻 */
   time_t st_ctime; /* 最後に変更したときの時刻 */
};
```

この構造体の各メンバは、普通、mknod()、utime()、read()、write()、truncate()などを呼び出したときに変更されます。

● エラーコード

EACCES、EBADF、EFAULT、ELOOP、ENAMETOOLONG、ENOENT、ENOMEM、ENOTDIR

● 互換性

ファイルの属性は、ファイルシステムによって異なります。そのため、stat構造体のst_blksizeやst_blocks、時間フィールドなどは、ファイルシステムによっては実装していません。

● 参照→chmod()、chown()、utime()

86 ファイルの状態を得る _fstat()

ヘッダー	sys/stat.h、sys/types.h
書式	int _fstat(int *fd*, struct _stat **buf*);
引数	*fd* ………ファイルデスクリプタ（普通はopen()が返した値）。 *buf* ……ファイルの状態を示す情報を保存する_stat構造体変数のポインタ。
戻り値	ファイルに関する情報の検索に成功したときは0、エラーの場合は-1。

解説

指定されたファイルに関する情報を取得します。

この関数のファミリーとして、以下の関数があります。

```
int _fstat32( int fd, struct __stat32 *buffer );
int _fstat64( int fd, struct __stat64 *buffer );
int _fstati64( int fd, struct _stati64 *buffer );
int _fstat32i64( int fd, struct _stat32i64 buffer );
int _fstat64i32( int fd, struct _stat64i32 *buffer );
```

互換性… Windows

参照→fstat()

87 現在のファイルポインタの位置を取得する ftell()

ヘッダ	stdio.h
書式	long ftell(FILE * *fp*);
引数	*fp* ………開いているストリーム（通常はfopen()が返した値）。
戻り値	成功したときは現在のファイルポインタの位置、失敗したときは-1。

解説

ファイル*fp*のファイルポインタの位置をバイト単位の値で返します。

注意

この関数によって返される値はlongですが、現在ではかなり大きなサイズのファイルを扱うことがあり、ファイルサイズが大きくてlongの範囲を超えたときの動作は保証されません。

参照→fseek()、fgetpos()、fsetpos()

例

fseek()の例参照。

88	日付と時間を返す ftime()
ヘッダ	sys/timeb.h
書式	int ftime(struct timeb *tp);
引数	*tp* ………時刻を保存するtimeb構造体のポインタ。
戻り値	常に0。

● 解説

現在の日付と時間をtimeb構造体のポインタ*tp*に保存します。この構造体は以下のように定義されています。

```
struct timeb {
   time_t time;
   unsigned short millitm;
   short timezone;
   short dstflag;
};
```

● 互換性

Windowsではvoid _ftime(struct _timeb *tp);を使います。Windowsの_ftime()は値を返しません。

89	日付と時間を返す _ftime()
ヘッダ	sys/types.h および sys/timeb.h
書式	void _ftime(struct _timeb *tp);
引数	*tp* ………時刻を保存する_timeb構造体のポインタ。

● 解説

現在の日付と時間を_timeb構造体のポインタ_tpに保存します。

● 参照→ftime()

90	データを書式化しないでファイルに書き込む fwrite()
ヘッダ	stdio.h
書式	size_t fwrite(const void **buf*, size_t *size*, size_t *n*,FILE **fp*);
引数	*buf* ……データを保存する変数のポインタ。 *size* ……書き込むデータ1つのサイズ（バイト単位）。 *n* ………書き込むデータの数。 *fp* ………書き込むファイル。
戻り値	書き込みに成功したデータの数。

解説

バッファ*buf*の中にある、1個のサイズが*size*バイトのデータを、ファイル*fp*に*n*個書き込みます。

例

次の例は、fwrite()とfread()を使ってファイルにアクセスする例です。

```
#include <stdio.h>

#define INFNAME "./test.dat"
#define OUTFNAME "./test.out"

int main(int argc, char *argv[])
{
   FILE *stream;
   char buff[256];
   int l, n;
   if((stream = fopen(INFNAME, "r")) == NULL)
   {
      printf("入力ファイルを開くことができません.¥n");
      return ;
   }

   /* 1バイトのデータを6個読み出す */
   n = fread(buff, sizeof(char), 6, stream);
   if (n != 6)
   {
      printf("正しく読み出せませんでした.¥n");
      fclose(stream);
      return;
   }
```

```
    else
        buff[n] = 0;
    printf("データ=%s¥n", buff);
    fclose(stream);

    if((stream = fopen(OUTFNAME, "w")) == NULL)
    {
        printf("出力ファイルを開くことができません.¥n");
        return ;
    }
    l = strlen(buff);

    /* ストリームに文字列を書き込む */
    if (l != fwrite(buff, sizeof(char), l, stream))
    {
        printf("すべてのデータを書き込めませんでした.¥n");
        printf("%dバイト書き込みました.¥n", l);
    }
    fclose(stream);

    return 0;
}
```


91 文字をファイルから読み出す getc()

ヘッダ	stdio.h
書式	int getc(FILE **fp*);
引数	*fp* ………開いているストリームのポインタ。
戻り値	成功したときは読み出した文字をintにキャストした値、エラーが発生したかファイルの終りに達したときはEOF。

● 解説

ファイル*fp*から次の文字をunsigned charとして読み出し、intにキャストして返します。機能としてはfgetc()とgetc()は同じですが、getc()はマクロとして実装されていることがあります。

● 例

次の例は、カレントディレクトリにあるファイルtest.txtを1バイトずつ読み込んでは標準出力に出力するという作業をファイルの最後まで繰り返す例です。

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   FILE *stream;
   int c;
   stream = fopen("./test.txt", "r");
   if (stream == NULL) {
      fprintf(stdout, "ファイルを開くことができません.¥n");
      return;
   }
   while (!feof(stream))
   {
      c = getc(stream);
      fputc(c, stdout);
   }
   fclose(stream);

   return 0;
}
```

● **参照**→ferror()、fgetc()、fopen()、fread()、fseek()、puts()、read()、scanf()、write()

92 文字を標準入力から読み出す getchar()

ヘッダ	stdio.h
書式	int getchar(void);
戻り値	成功したときは読み出した文字をintにキャストした値。 エラーが発生したかファイルの終りに達したときはEOF。

● **解説**

標準入力(stdin)から文字を読み出します。機能としてはgetc(stdin)と同じです。

● **例**

次の例は、標準入力から文字を1バイトずつ読み込んで、改行(¥n)を読み込んだら文字列として標準出力に書き出します。

```
#include <stdio.h>
#define BUFFSIZE 1024

int main(int argc, char *argv[])
{
   int c, i;
   char buff[BUFFSIZE];
   i = 0;
   while (!feof(stdin))
   {
      c = getchar();
      buff[i++] = c;
      if ((c == '¥n') || i == BUFFSIZE-1)
      {
         buff[i] = 0;
         puts(buff);
         i = 0;
      }
   }
}
```

● **参照**→ferror()、fopen()、fread()、fseek()、puts()、read()、scanf()、write()

93 カレントディレクトリ名を取得する getcwd()

ヘッダ	unistd.h
書式	char *getcwd(char **buf*, size_t *size*);
引数	*buf* ……ディレクトリパスを保存するバッファのアドレス。NULLも指定可。 *size* ……バッファの長さ。
戻り値	成功したときは取得した絶対パスのポインタ。 エラーが発生したときはNULL。

● **解説**

カレントディレクトリの絶対パス名をバッファ*buf*に取得します。バッファのサイズにNULLを指定すると、サイズが*size*であるバッファが自動的に割り当てられます。

● **例**… 次の例はカレントディレクトリを調べて表示する例です。

```
#include <stdio.h>
#ifdef WIN32
#include <direct.h>
#else
#include <unistd.h>
#endif
#include <stdlib.h>

#define BUFFSIZE 1024

int main(int argc, char *argv[])
{
   char buff[BUFFSIZE];
#ifdef WIN32
   _getcwd(buff, BUFFSIZE);
   /* あるいは次のようにバッファにNULLを
    * 指定することもできる
    * strcpy(buff, _getcwd(NULL, BUFFSIZE)); */
#else
   getcwd(buff, BUFFSIZE);
#endif
   puts(buff);

   return 0;
}
```

● 参照→chdir()

94 カレントディレクトリ名を取得する _getcwd()

ヘッダ	direct.h
書式	char *getcwd(char **buf*, size_t *size*);
引数	*buf* ……ディレクトリパスを保存するバッファのアドレス。NULLも指定可。 *size* ……バッファの長さ。
戻り値	成功したときは取得した絶対パスのポインタ。 エラーが発生したときはNULL。

● 解説

カレントディレクトリの絶対パス名をバッファ*buf*に取得します。

● **準拠**… ISO C++

● **参照**→getcwd()

95 グループIDを得る getegid()

ヘッダ	unistd.h、sys/types.h
書式	gid_t getegid(void);
戻り値	実効グループID。

● **解説**

現在のプロセスの実効グループIDを返します。

● **準拠**… POSIX

● **参照**→getgid()

96 環境変数を取得する getenv()

ヘッダ	stdlib.h
書式	char *getenv(const char **name*);
引数	*name* …取得したい環境変数名。
戻り値	環境文字列のポインタ。文字列が存在しないときはNULL。

● **解説**

指定した文字列と一致する文字列を環境変数のリストから探して、その値を返します。この関数のワイド文字バージョンは_wgetenv()です。

● **準拠**… POSIX

● **例**

次の例は、プログラム起動時の引数で指定された環境変数の値を出力するプログラムの例です。

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[ ])
{
   char *envvar;
   int i;
   if (argc < 2)
   {
      printf("使い方:getenv 環境変数名¥n");
      exit (1);
   }
   for (i=1; i<argc; i++)
   {
      envvar = getenv(argv[i]);
      if (envvar != NULL)
         printf("環境変数%s の内容: %s¥n", argv[i], envvar);
      else
         printf("環境変数%s は設定されていません.¥n");
   }
   return 0;
}
```

● **参照**→putenv()、setenv()、unsetenv()

97	グループIDを得る getgid()
ヘッダ	unistd.h、sys/types.h
書式	gid_t getgid(void);

● **解説**

現在のプロセスの実グループIDを返します。

● **準拠**… POSIX

● **参照**→getegid()

98	ファイルシステム記述ファイルの情報を取得する getmntent()
ヘッダ	stdio.h、mntent.h
書式	struct mntent *getmntent(FILE **fp*);
引数	*fp* ………ファイルシステム記述ファイルのファイルポインタ。
戻り値	成功したときはmntent構造体のポインタ、失敗したときはNULL。

● **解説**

ファイルシステム記述ファイルの1行ぶんの情報を取得します。ファイルシステム記述ファイル全体の情報を取得するためには、この関数がNULLを返すまで繰り返し呼び出します。*fp*はsetmntent()を呼び出して取得します。

● **互換性**… Windowsではシステム記述ファイルを使わないのでこの関数は使いません。

● **例**

この例は、/etc/fstabと/etc/mtabのすべての内容を表示します。ファイル全体の情報を取得するために、NULLを返すまでこの関数を繰り返し呼び出している点に注意してください。

```
#include <stdio.h>
#include <mntent.h>

void printmnt(const char *fname)
{
   FILE *filep;
   struct mntent *pmnt;
   filep = setmntent(fname, "ro");
   if (filep == NULL)
   {
      fprintf(stderr, "ファイル%s を開けません.¥n", fname);
      return;
   }
   printf("%s:¥nmnt_fsname mnt_dir mnt_type ", fname);
   printf("mnt_opts mnt_freq mnt_passno¥n");
   while((pmnt = getmntent(filep)) != NULL)
   {
      if (pmnt == NULL) break;
      printf("%-12s%-12s%-12s%-22s %d %d¥n",
         pmnt->mnt_fsname, /* マウントされているファイルシステムの名前 */
         pmnt->mnt_dir, /* ファイルシステムのパスプリフィックス */
         pmnt->mnt_type, /* マウントタイプ (mntent.h参照) */
         pmnt->mnt_opts, /* マウントオプション (mntent.h参照) */
         pmnt->mnt_freq, /* ダンプするかどうか */
         pmnt->mnt_passno); /* ファイルチェックするかどうか */
```

```
    }
    endmntent(filep);
}

int main(int argc, char *argv[])
{
    printmnt("/etc/fstab");
    printmnt("/etc/mtab");

    return 0;
}
```

● **関連ファイル**

/etc/fstab ……ファイルシステム記述ファイル

/etc/mtab ……マウントされているファイルシステム記述ファイル

● **参照**→fopen()、setmntent()

99	プロセスIDを得る getpid()
ヘッダ	unistd.h
書式	pid_t getpid(void);

● **解説**

現在のプロセスIDを返します。

● **準拠**… POSIX

100	プロセスIDを得る _getpid()
ヘッダ	process.h
書式	int _getpid(void);

● **解説**

プロセスIDを取得します。

● **準拠**… ISO C++

101	親プロセスIDを得る getppid()
ヘッダ	unistd.h
書式	pid_t getppid(void);

●解説

現在のプロセスの親プロセスのプロセスIDを返します。

102	標準入力から1行の文字列を読み出す gets()
ヘッダ	stdio.h
書式	char *gets(char **buf*);
引数	*buf* ……読み出した文字列を保存するバッファのポインタ
戻り値	成功したときはバッファを指すポインタ。 ファイルの終わりを検出したかエラーの場合NULL。

●解説

標準入力（stdin）から改行文字かEOFまで読み出してバッファ*buf*に保存します。文字列の最後の改行は¥0に置き換えられます。読み込み文字列の長さはチェックされないので、バッファより長い文字列を読み込む可能性があり、そのときの動作は保証されません。

●注意

読み込み文字列の長さがチェックされないことを利用して本来アクセスすべきでないメモリにアクセスしてセキュリティに重大な影響を与える可能性もあります。そのため、バッファの長さより長い文字列を読み込む可能性がある場合やセキュリティの問題が重要な場合は、fgets()を使うべきです。

●例

次の例では、名前を文字列として読み出します。

```
#include <stdio.h>
#include <locale.h>

int main(int argc, char *argv[])
{
   char urname[256];

   printf("名前は？>");
```

```
    gets(urname);

    /* 上の行の代わりに次のようなコードを使うべき
     * fgets(urname, 256, stdin);
     * urname[strlen(urname)-1] = '¥0'; */

    printf("Hello %s! 元気でやってるかい?¥n", urname);

    return 0;
}
```

103 カレンダー時刻を万国標準時に変換する gmtime()

ヘッダ	time.h
書式	struct tm *gmtime(const time_t **timep*);
引数	*timep* …時刻データが入っているtime_t構造体のポインタ。

● 解説

time_t構造体に入っているカレンダー時刻を、tm構造体形式の万国標準時（UTC、世界協定時刻）に変換します。

● 準拠… ANSI

● 例

asctime()の例参照。

104 ファイルシステム記述ファイルのオプションを調べる hasmntopt()

ヘッダ	stdio.h、mntent.h
書式	char *hasmntopt(const struct mntent **mnt*, const char **opt*);
引数	*mnt* ……調べるmntent構造体のポインタ。 *opt* ……オプション文字列。
戻り値	オプション文字列があればその文字列のポインタ、そうでなければNULL。

● 解説

mntent構造体のmnt_optsに指定したオプション文字列があるかどうか調べます。

03-03 関数の説明

● 仕様… BSD

● 例

次の例は、ファイル/etc/fstabの中でオプションに"user"が含まれる行を表示します。

```
#include <stdio.h>
#include <mntent.h>

int main(int argc, char *argv[])
{
   FILE *filep;
   struct mntent *pmnt;
   filep = setmntent("/etc/fstab", "ro");
   printf("%s:¥nmnt_fsname mnt_dir mnt_type ", fname);
   printf("mnt_opts mnt_freq mnt_passno¥n");
   while((pmnt = getmntent(filep)) != NULL)
   {
      if (pmnt == NULL) break;
      /* オプションに"user"が含まれるエントリーを表示する */
      if (hasmntopt(pmnt, "user"))
         printf("%-12s%-12s%-12s%-22s %d %d¥n",
            pmnt->mnt_fsname, /* マウントされているファイルシステムの名前 */
            pmnt->mnt_dir, /* ファイルシステムのパスプリフィックス */
            pmnt->mnt_type, /* マウントタイプ (mntent.h参照) */
            pmnt->mnt_opts, /* マウントオプション (mntent.h参照) */
            pmnt->mnt_freq, /* ダンプするかどうか */
            pmnt->mnt_passno); /* ファイルチェックするかどうか */
      }
   endmntent(filep);
}
```

● 関連ファイル

/etc/fstab……ファイルシステム記述ファイル

/etc/mtab……マウントされたファイルシステムの記述ファイル

● 互換性

Windowsにはこの関数はありません。

● 参照→fopen()、getmntent()、setmntent()

105 文字がアルファベットまたは数字であるかどうか調べる isalnum()

ヘッダ	ctype.h
書式	int isalnum (int *c*);
引数	*c* ………調べる文字。
戻り値	文字cが調べた文字の種類に合っていれば0以外、そうでなければ0。

● 解説

*c*をチェックしてアルファベットまたは数字であるか調べた結果を返します。

isalnum()は(isalpha (c) || isdigit(c))と同じです。結果は現在のlocaleに従って変わります。*c*はunsigned charかEOFでなければなりません。

● 準拠… ANSI

● 例

次の例は、ユーザーが入力した文字の種類を出力するプログラムの例です。

```
#include <stdio.h>
#include <locale.h>
#include <ctype.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ];
   int c;
   while (1)
   {
      printf("文字>");
      gets(buff);
      c = (int)buff[0];
      printf("文字%c (%x) は", c, c);
      if (isalnum(c))
         printf("アルファベットまたは数字です。¥n");
      if (isalpha(c))
         printf("アルファベットです。¥n");
      if (iscntrl(c))
         printf("制御文字です。¥n");
      if (isdigit(c))
         printf("数値のための文字です。¥n");
      if (isgraph(c))
         printf("スペースを除いた表示可能な文字です。¥n");
      if (islower(c))
```

```
            printf("小文字です。¥n");
        if (isprint(c))
            printf("表示可能です。¥n");
        if (ispunct(c))
            printf("区切り文字です。¥n");
        if (isspace(c))
            printf("空白文字です。¥n");
        if (isupper(c))
            printf("大文字です。¥n");
        if (isxdigit(c))
            printf("16進数の数字です。¥n");
    }
    return 0;
}
```

● **参照**→setlocale()、tolower()、toupper()

106 文字がアルファベットであるかどうか調べる isalpha()

ヘッダ	ctype.h
書式	int isalpha (int *c*);
引数	*c* ………調べる文字。
戻り値	文字cがアルファベットの文字であれば0以外、そうでなければ0。

● **解説**

*c*をチェックして、現在のlocaleに従ってアルファベットかどうか調べます。結果は現在のlocaleに従って変わります。*c*はunsigned charかEOFでなければなりません。

● **例**

isalnum()の例参照。

● **参照**→setlocale()、tolower()、toupper()

107	デスクリプタが端末かどうか調べる **isatty()**
ヘッダ	io.h
書式	int isatty (int *desc*);
引数	*desc* ……ファイルデスクリプタ。
戻り値	端末をオープンしたなら1、そうでなければ0。

● 解説

デスクリプタ*desc*がそのシステムの端末あるいはキャラクタデバイス（端末、コンソール、プリンタ、シリアルポートなど）に結合しているかどうか調べます。

108	デスクリプタが端末かどうか調べる **_isatty()**
ヘッダ	io.h
書式	int _isatty (int *desc*);
引数	*desc* ……ファイルデスクリプタ。
戻り値	端末をオープンしたなら1、そうでなければ0。

● 解説

デスクリプタ*desc*がそのシステムの端末あるいはキャラクタデバイス（端末、コンソール、プリンタ、シリアルポートなど）に結合しているかどうか調べます。

● 例

次の例は、stdoutが標準出力（コンソール）に設定されているか、ファイルにリダイレクトされているか調べるプログラムの例です。

```
#include <io.h>
#include <stdio.h>

int main( void )
{
   if( _isatty( _fileno( stdout ) ) )
      printf( "stdoutは標準出力¥n" );
   else
      printf( "stdoutはファイルにリダイレクト¥n");

   return 0;
}
```

109 文字が制御文字であるかどうか調べる iscntrl()

ヘッダ	ctype.h
書式	int iscntrl (int *c*);
引数	*c* ………調べる文字。
戻り値	文字が制御文字であれば0以外、そうでなければ0。

● 解説

文字*c*が制御文字であるかどうか調べます。*c*はunsigned charの文字かEOFでなければなりません。結果は現在のlocaleに従って変わります。

● 例… isalnum()の例参照。

110 文字が数値のための文字であるかどうか調べる isdigit()

ヘッダ	ctype.h
書式	int isdigit (int *c*);
引数	*c* ………調べる文字。
戻り値	文字*c*が数字の文字であれば0以外、そうでなければ0。

● 解説

*c*を数字（0〜9まで）かどうか調べます。*c*はunsigned charかEOFでなければなりません。結果は現在のlocaleに従って変わります。

● 例… isalnum()の例参照。

111 文字がスペースを除く表示可能な文字であるかどうか調べる isgraph()

ヘッダ	ctype.h
書式	int isgraph (int *c*);
引数	*c* ………調べる文字。
戻り値	文字*c*が表示可能な文字であれば0以外、そうでなければ0。

● 解説

文字*c*が表示可能な文字かどうかを調べます。スペースは表示可能な文字とみなされません。*c*はunsigned charかEOFでなければなりません。結果は現在のlocaleに従って変わります。

● 例

isalnum()の例参照。

112 文字が小文字であるかどうか調べる islower()

ヘッダ	ctype.h
書式	int islower (int *c*);
引数	*c* ………調べる文字。
戻り値	文字*c*が小文字であれば0以外、そうでなければ0。

● 解説

文字*c*が小文字であるかどうか調べます。*c*はunsigned charかEOFでなければなりません。結果は現在のlocaleに従って変わります。

● 例

次の例は、入力された文字が小文字であるかどうか調べて、その結果を出力します。

```
#include <stdio.h>

int main( void )
{
   char c;

   printf("文字>");
   scanf("%c", &c);

   if ( islower(c) )
      printf( "%cは小文字¥n", c );
   else
      printf( "%cは大文字¥n", c );

   return 0;
}
```

● 参照→setlocale()、tolower()、toupper()

113 文字が表示可能であるかどうか調べる isprint()

ヘッダ	ctype.h
書式	int isprint (int *c*);
引数	*c* ………調べる文字。
戻り値	文字*c*が表示可能であれば0以外、そうでなければ0。

● 解説

文字*c*が表示可能かどうか調べます。表示可能な文字にはスペースを含みます。*c*はunsigned charかEOFでなければなりません。結果は現在のlocaleに従って変わります。

● 例

isalnum()の例参照。

114 文字が区切り文字であるかどうか調べる ispunct()

ヘッダ	ctype.h
書式	int ispunct (int *c*);
引数	*c* ………調べる文字。
戻り値	文字*c*が区切り文字であれば0以外、そうでなければ0。

● 解説

文字*c*が区切り文字かどうかを調べます。区切り文字には、たとえば、, .; : などが含まれますが、スペースと英数字は除かれます。*c*はunsigned charの文字かEOFでなければなりません。結果は現在のlocaleに従って変わります。

● 例… isalnum()の例参照。

115 文字が空白文字であるかどうか調べる isspace()

ヘッダ	ctype.h
書式	int isspace (int *c*);
引数	*c* ………調べる文字。
戻り値	文字*c*が空白文字であれば0以外、そうでなければ0。

● 解説

文字*c*が空白文字かどうか調べます。*c*はunsigned charかEOFでなければなりません。結

果は現在のlocaleに従って異なります。たとえば、localeがCかPOSIXの場合、空白文字は、スペース、フォームフィード（¥f）、改行（¥n）、復帰（¥r）、水平タブ（¥t）、垂直タブ（¥v）です。

● **例**… isalnum()の例参照。

116 文字が大文字であるかどうか調べる isupper()

ヘッダ	ctype.h
書式	int isupper (int *c*);
引数	c ………調べる文字。
戻り値	文字*c*が大文字であれば0以外、そうでなければ0。

● **解説**

文字*c*が大文字であるかどうか調べます。*c*はunsigned charかEOFでなければなりません。結果は現在のlocaleに従って異なります。

● **例**… isalnum()の例参照。

117 文字が16進数の数字文字であるかどうか調べる isxdigit()

ヘッダ	ctype.h
書式	int isxdigit (int *c*);
引数	c ……調べる文字。unsigned charの文字かEOFでなければなりません。
戻り値	文字*c*が16進数の数字文字であれば0以外、そうでなければ0。

● **解説**

文字*c*が16進数の数字文字かどうか調べます。16進数の数字文字は、0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E Fのいずれかです。*c*はunsigned charかEOFでなければなりません。結果は現在のlocaleに従って変わります。

● **例**

isalnum()の例参照。

118 ロング整数の絶対値を計算する labs()

ヘッダ	stdlib.h
書式	long int labs(long int *x*);
引数	*x* ………絶対値を計算するロング整数。
戻り値	ロング整数の引数の絶対値。

● 解説

整数の引数*x*の絶対値を計算します。

● 注意

得ることができる最も小さい負の整数の絶対値の大きさは定義されていません。

● 例

abs()の例参照。

● 備考

long long型が定義されている場合、long long intの絶対値を計算する関数llabs()も使えることがあります。llabs()とlabs()は引数と戻り値の型が異なるだけで、llabs()の使い方はlabs()と同じです。

119 仮数と指数から実数を計算する ldexp()

ヘッダ	math.h
書式	double ldexp(double *x*, int *e*);
引数	*e* ………2のe乗を計算するときのeの整数値 *x* ………pow(2, e)に掛ける浮動小数点の値

● 解説

2をe乗して*x*倍した値(pow(2, e) * x)を計算します。

● 例

次の例は、数値のn乗のm倍を計算する例です。

```
#include <math.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   float f;
   double x, y;
   int e;

   printf("(2 ^ e) * x の計算¥n");

   printf("e>");
   scanf("%d", &e);

   printf("実数x>");
   scanf("%f", &f);

   x = (double)f;
   y = ldexp(x, e);

   printf("2の%d乗の%5.2f倍=%2.1f¥n", e, x, y);
   printf("2の%d乗の%5.2f倍=%2.1f¥n", e, x, (pow(2, e) * x));
}
```

● **参照**→frexp()、pow()

120　ロング整数の割り算の商と余りを計算する ldiv()

ヘッダ	stdlib.h
書式	ldiv_t ldiv(long int *n*, long int *d*);
引数	*n* ………割られる値 *d* ………割る値
戻り値	ldiv_t構造体。

● **解説**

n/dの値を計算してldiv_t構造体に保存します。ldiv_t構造体は次のとおりです。

```
typedef struct _ldiv_t {
   long quot;    /* 商（quotient） */
   long rem;     /* 余り（remainder） */
} ldiv_t;
```

● **参照**→div()

121 時刻の値を現地時間に変換する localtime()

ヘッダ	time.h
書式	struct tm *localtime(const time_t **timep*);
引数	*timep* …time_t構造体のポインタ。
戻り値	現地時間が入ったtm構造体のポインタ。

● 解説

time_t構造体に入っている時刻を現地時間に変換してtm構造体のポインタとして返します。

● 互換性

この関数はANSI関数ですが、実装の詳細はプラットフォームによって異なります。

● 例

次の例は、日本における現在の日時を出力する例です。

```
#include <time.h>
#include <stdio.h>
#include <locale.h>

int main(void)
{
   time_t timer;
   struct tm *ptm;

   setlocale(LC_ALL, "japanese");

   /* 現在時刻を取得する */
   timer = time(NULL);

   /* 日付/時刻を構造体に変換する */
   ptm = localtime(&timer);

   printf("現在の日時（日本）=%s", asctime(ptm));

   return 0;
}
```

122	自然対数を計算する log()
ヘッダ	math.h
書式	double log(double *x*);
引数	*x* ………自然対数を計算する値。
戻り値	*x*の自然対数の値。

● 解説

*x*の自然対数を計算して返します。

● エラーコード… EDOM、ERANGE

123	常用対数を計算する log10()
ヘッダ	math.h
書式	double log10(double *x*);
引数	*x* ………常用対数を計算する値。
戻り値	*x*の常用対数の値。

● 解説

*x*の常用対数を返します。

● エラーコード… EDOM、ERANGE

124	環境を復元する longjmp()
ヘッダ	setjmp.h
書式	void longjmp(jmp_buf *env*, int *val*);
引数	*env* ……環境を保存したjmp_buf構造体変数。 *val*………setjmp()が返す値。

● 解説

最後にsetjmp(env)を呼び出したときの状況を復元します。

関数setjmp()を呼び出すと、現在の環境が保存されます。次に、longjmp()を呼び出すと、保存されている環境が復元され、対応するsetjmp()が実行されてvalueを返したのと同じ状態で、setjmp()の直後の位置に制御が戻ります。プログラムで低レベルなエラーや割り込みが発生したときの処理にsetjmp()とlongjmp()を組み合わせて使うことがよくあります。valに0を指定した場合、setjmp()は0を返す代わりに1を返します。

この関数は呼び出し側にリターンしません。

● **互換性**

関数setjmp()が保存する環境値やこの関数が復元する環境値は、実行環境によって異なります。

● **参照**→setjmp()

125 ファイルの読み書きオフセットの位置を変える lseek()

ヘッダ	sys/types.h、unistd.h
書式	long lseek(int *fd*, off_t *offset*, int *origin*);
引数	*fd* ………ファイルデスクリプタ。 *offset* …移動する量。*origin*からのオフセット（バイト単位）。 *origin* …初期位置（fseek()に指定可能な値のいずれか）。
戻り値	成功したときは移動後のファイル位置（ファイルの先頭からのバイト数）、失敗したときは-1。

● **解説**

ファイルデスクリプタ*fd*で識別されるファイルの現在のファイル位置を示すポインタの位置を、*origin*で指定された位置から*offset*バイトの位置に移動します。

● **エラーコード**… EBADF、EINVAL、ESPIPE

● **例**

fseek()の例参照。

● **参照**→fseek()

126 ファイルの状態を取得する lstat()

ヘッダ	sys/stat.h、unistd.h
書式	int lstat(const char **fname*, struct stat **buf*);
引数	*fname* …情報を取得するファイル名。 *buf* ……取得した情報を保存するstat構造体のポインタ。
戻り値	成功したときは0、エラーの場合は-1（エラーコードがerrnoにセットされます）。

解説

指定されたファイル*fname*に関する情報を取得します。lstat()はstat()と同様に情報を取得しますが、リンクを追跡して得られるファイルではなくリンク自身の状態を取得するという点で違います。

stat構造体は次のように定義されています。

```
struct stat {
    dev_t        st_dev;      /* デバイス */
    ino_t        st_ino;      /* inode */
    mode_t       st_mode;     /* 保護モード */
    nlink_t      st_nlink;    /* ハードリンクの数 */
    uid_t        st_uid;      /* 所有者のユーザID */
    gid_t        st_gid;      /* 所有者のグループID */
    dev_t        st_rdev;     /* デバイスタイプ（inodeデバイスでない場合）*/
    off_t        st_size;     /* サイズ（バイト単位） */
    blksize_t    st_blksize;  /* ブロックサイズ */
    blkcnt_t     st_blocks;   /* ブロック数 */
    time_t       st_atime;    /* 最終アクセス日時 */
    time_t       st_mtime;    /* 最終更新日時 */
    time_t       st_ctime;    /* 最終ステータス変更日時 */
};
```

エラーコード

EACCES、EBADF、EFAULT、ELOOP、ENAMETOOLONG、ENOENT、ENOMEM、ENOTDIR

互換性

この関数の実装の詳細はファイルシステムに依存します。Windowsでファイルの状態に関する情報を取得するときには、_stat()、_wstat()、_stati64()、_wstati64()を使います。

参照→chmod()、fstat()

127 メモリを動的に割り当てる malloc()

ヘッダ	stdlib.h
書式	void *malloc(size_t *size*);
引数	*size* ……確保するメモリのサイズ（バイト単位）。
戻り値	割り当てられたメモリのポインタ、失敗したときはNULL。

●解説

システムのメモリから*size*バイトのメモリを確保してプログラムに割り当て、割り当てられたメモリのポインタを返します。メモリはどの種類の変数にも適するようにアラインメントされていて、通常、希望する型にキャストして使います。

メモリの内容が初期化（クリア）されるわけではありません。割り当てられたメモリの内容を0で初期化したいときには、memset()を呼び出して初期化する必要があります。

●互換性

環境変数の設定や特定の関数の呼び出しでmalloc()の動作を制御できるような実装がされている場合があります。また、デバッグバージョンのライブラリとリリースバージョンのライブラリで動作の詳細が異なることがあります。

●例

次の例は動的に確保したメモリを使う方法を示すプログラムの例です。

```
#include <memory.h>
#include <string.h>
#include <stdio.h>

#ifdef WIN32
#include <malloc.h>
#endif

int main(int argc, char *argv[])
{
   char *ps;
   char buff[256];
   int c;

   printf("文字列>");
   scanf("%s", buff);
   ps = malloc(128);

   if (ps == NULL)
   {
      printf("メモリを確保できませんでした.¥n");
      return;
   }

   /* メモリを0で初期化する */
   memset(ps, 0, 128);
```

```
    /* 128バイトまたは'x'までコピーする */
    c = 'x';
    memccpy(ps, buff, c, 128);
    printf("buff=%s¥nps=%s¥n", buff, ps);

    /* メモリを解放する */
    free(ps);

    return 0;
}
```

128 マルチバイト文字の長さを取得する _mbclen()

ヘッダ	mbstring.h
書式	size_t _mbclen(const unsigned char *c);
引数	*c* ………調べるマルチバイト文字のポインタ。
戻り値	成功したときは1文字あたりのバイト数、文字が無効か0、あるいはNULLのときは-1。

● 解説

文字*c*をスキャンして、1文字あたりのバイト数を返します。

129 大文字を小文字にする _mbctolower()

ヘッダ	mbstring.h
書式	unsigned int _mbctolower(unsigned int *c*);
引数	*c* ………小文字に変換する文字。
戻り値	変換できれば変換後の文字、できなければ変換前の文字。

● 解説

マルチバイト文字*c*が小文字にできる文字であるならば小文字に変換します。

● 参照→tolower()

130 1文字のバイト数を求める mblen()

ヘッダ	stdlib.h
書式	int mblen(const char **s*, size_t *n*);
引数	*s* ………調べる文字が入っている文字列のポインタ。 *n* ………調べるバイト数。
戻り値	成功したときは1文字あたりのバイト数、文字が無効か0、あるいはsがNULLのときは-1。

● 解説

文字列*s*の最初の*n*バイトをスキャンして、マルチバイト文字の1文字あたりのバイト数を返します。*n*にMB_CUR_MAXを指定すると、現在のlocaleで最も長いマルチバイト文字列の長さが指定されます。

● 参照→_mbclen()

131 文字列に一連の文字のいずれかの文字があるか調べる _mbspbrk()

ヘッダ	mbstring.h
書式	unsigned char *_mbspbrk(const unsigned char **str1*, const unsigned char **str2*);
引数	*str1* ……*str2*の一連の文字のいずれかが含まれているかどうかを調べる文字列。 *str2* ……含まれるかどうか検査する文字を保存したNULLで終わるキャラクタ配列。
戻り値	文字列*str2*のいずれかの文字が*str1*に現れた位置。

● 解説

*str2*で指定した一連の文字のいずれかの文字が*str1*に含まれているかどうかを調べます。

● 参照→strpbrk()

132 マルチバイト文字列をワイド文字列に変換する
mbstowcs()

ヘッダ	stdlib.h
書式	size_t mbstowcs(wchar_t **pwcs*, const char **pmbc*, size_t *n*);
引数	*pwcs*……ワイド文字列を保存するバッファのポインタ。 *pmbc* …マルチバイト文字列が入っているバッファのポインタ。 *n* ………変換する文字数（マルチバイト文字の文字数で指定）。
戻り値	変換されたワイド文字数。 マルチバイト文字列に無効なマルチバイト文字が含まれるときは-1。

●解説

マルチバイト文字列*pmbc*の先頭から*n*文字をワイド文字列に変換して*pwcs*に保存します。

●例

次の例は、マルチバイト文字列をいったんワイド文字列に変換してから、再びマルチバイト文字列に変換するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int main(void)
{
    char mbc [] = "マルチバイト文字列";
    char mbs[256];
    wchar_t wcs[100];
    size_t size, n;

    setlocale(LC_ALL, "japanese");

    /* 必要な長さを求める */
    size = mbstowcs(NULL, mbc, 0);
    printf("wcsに変換前(%d)=%s¥n", size, mbc);

    /* ワイド文字列に変換する */
    n = mbstowcs(wcs, mbc, size);
    wcs[n] = NULL;

    wprintf(L"wcsに変換後(%d)=%s¥n", n, wcs);

    /* マルチバイト文字列に変換する */
    size = wcstombs(mbs, wcs, 256);
    mbs[size] = 0;

    printf("mbsに変換後(%d)=%s¥n", size, mbs);

    return 0;
}
```

● 参照→mblen()、mbtowc()、wcstombs()、wctomb()

133 マルチバイト文字をワイド文字に変換する mbtowc()

ヘッダ	stdlib.h
書式	int mbtowc(wchar_t **pwc*, const char **pmbc*, size_t *n*);
引数	*pwc* ……ワイド文字を保存するバッファのポインタ。 *pmbc* …マルチバイト文字が入っているバッファのポインタ。 *n* ………変換するバイト数。
戻り値	マルチバイト文字のバイト数。マルチバイト文字が有効でないときは-1。

● 解説

マルチバイト文字*pmbc*をワイド文字*pwc*に変換します。変換する前に最大*n*バイトまでをチェックして変換が実行されます。

● 例

次の例は、マルチバイト文字をワイド文字に変換してから、再びマルチバイト文字に変換する例です。

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int main(void)
{
    char mbc [] = "あ";
    char mbs[256];
    wchar_t wcc;
    size_t size;

    setlocale(LC_ALL, "japanese");

    mbtowc(&wcc, mbc, 2);

    wprintf(L"wcに変換後=%c¥n", wcc);

    size = wctomb(mbs, wcc);
    mbs[size] = 0;

    printf("mbに変換後=%s¥n", mbs);

    return 0;
}
```

● **参照**→mblen()、mbstowcs()、wcstombs()、wctomb()

134 メモリ領域をコピーする memccpy()

ヘッダ	string.h
書式	void *memccpy(void **dest*, const void **src*, int *c*, size_t *n*);
引数	*dest* ……コピーした結果を保存するところのポインタ。 *src*………コピー元のメモリ領域のポインタ。 *c* ………遭遇したらコピーを中止する文字。 *n* ………コピーする最大のバイト数。
戻り値	*dest*の中に*c*があるときは*c*の次にあるキャラクタ型の変数を指すポインタ、見つからなかったときはNULL。

● **解説**

メモリ領域*src*からメモリ領域*dest*に最大で*n*バイトコピーします。*n*バイトコピーする前に文字*c*があれば、そこでコピーを中止します。

● **例**

malloc()の例参照。

● **参照**→memcpy()

135 メモリの中で特定の文字を探す memchr()

ヘッダ	string.h
書式	void *memchr(const void **pbuf*, int *c*, size_t *n*);
引数	*pbuf*……文字を探すメモリバッファのポインタ。 *c* ………探す文字。 *n* ………探す文字数。
戻り値	文字*c*があればその文字のポインタ、文字がない場合はNULL。

● **解説**

メモリ領域*pbuf*の最初の*n*バイトの部分に文字*c*があるかどうか調べます。

● **例**

次の例は、ユーザーが入力した文字列の中から、特定の文字を探すプログラムの例です。

```
#include <memory.h>
#include <string.h>
#include <stdio.h>
#ifdef WIN32
#include <malloc.h>
#endif

int main(int argc, char *argv[])
{
   char *pbuf;
   char c;

   /* 0で初期化したメモリを確保する */
   pbuf = (char *)calloc(128, sizeof(char));
   if (pbuf == NULL)
   {
      printf("メモリを確保できませんでした.¥n");
      return;
   }

   printf("文字列>");
   fgets(pbuf, 128, stdin);
   printf("文字列>%s¥n", pbuf);

   printf("探す文字>");
   scanf("%c", &c);

   if (memchr(pbuf, c, strlen(pbuf)) != NULL)
      printf("文字%cは存在します.¥n", c);
   else
      printf("文字%cは存在しません.¥n", c);

   /* メモリを解放する */
   free(pbuf);

   return 0;
}
```

136 メモリ領域を比較する memcmp()

ヘッダ	string.h
書式	int memcmp(const void **pbuf1*, const void **pbuf2*, size_t *n*);
引数	*pbuf1* …第1のメモリのポインタ。 *pbuf2* …第2のメモリのポインタ。 *n* ………比較する文字数。
戻り値	*pbuf1*の最初の*n*バイトが*pbuf2*の最初の*n*バイトよりも小さいときは負の整数、同じときは0、大きいときは正の整数。

● 解説

メモリ領域*pbuf1*と*pbuf2*の最初の*n*バイトを比較します。

● 例

次の例は、ユーザーが入力した文字列を比較するプログラムの例です。

```
#include <string.h>
#include <stdio.h>
#define BUFFSIZE 256

int main(int argc, char *argv[])
{
   char buf1[BUFFSIZE], buf2[BUFFSIZE];

   printf("文字列1>");
   fgets(buf1, BUFFSIZE, stdin);
   printf("文字列2>");
   fgets(buf2, BUFFSIZE, stdin);

   /* 最初の5バイトを比較する */
   if (memcmp(buf1, buf2, 5) == 0)
      printf("文字列は一致しています.¥n");
   else
      printf("文字列は一致していません.¥n");

   return 0;
}
```

137 メモリ領域をコピーする memcpy()

ヘッダ	string.h
書式	void *memcpy(void **dest*, const void **src*, size_t *n*);
引数	*dest* ……コピー先のメモリのポインタ。 *src*………コピー元のメモリのポインタ。 *n* ………比較する文字数。
戻り値	*dest*のポインタ。

●解説

メモリ領域*src*の先頭*n*バイトをメモリ領域*dest*にコピーします。

コピー元の領域とコピー先の領域が重なっている場合の動作は保証されません。メモリ領域が重なっている場合はmemmove()を使ってください。

●例

次の例は、ユーザーが入力した文字列のうち、最初の5バイトだけをコピーして出力する例です。

```
#include <string.h>
#include <stdio.h>
#define BUFFSIZE 256

int main(int argc, char *argv[])
{
   char buf1[BUFFSIZE], buf2[BUFFSIZE];
   printf("文字列>");
   fgets(buf1, BUFFSIZE, stdin);

   /* 最初の5バイトをコピーする */
   memcpy(buf2, buf1, 5);
   buf2[5] = 0;
   printf("%s¥n", buf2);

   return 0;
}
```

138 メモリ領域をコピーする memmove()

ヘッダ	string.h
書式	void *memmove(void **dest*, const void **src*, size_t *n*);
引数	*dest* ……コピー先のメモリのポインタ。 *src*………コピー元のメモリのポインタ。 *n* ………比較する文字数。
戻り値	*dest*のポインタ。

● 解説

メモリ領域*src*の先頭*n*バイトをメモリ領域*dest*にコピーします。コピー元とコピー先の領域が重なっていてもよいという点を除いてmemcpy()と同じです。

● 参照→memccpy()、memcpy()

139 メモリ領域を特定のバイトで埋める memset()

ヘッダ	string.h
書式	void *memset(void **s*, int *c*, size_t *n*);
引数	*s* ………メモリのポインタ。 *c* ………メモリにセットするバイト。 *n* ………cをセットするバイト数。
戻り値	*s*のポインタ。

● 解説

メモリ領域*s*の先頭から*n*バイトを*c*で埋めます。

● 例

malloc()の例参照。

140 ディレクトリを作成する mkdir()

ヘッダ	dir.h、direct.h
書式	int mkdir(const char **path*);
引数	*path*……作成するディレクトリのパス。
戻り値	新しいディレクトリが作成されたときは0、エラーのときは-1。

● 解説

ディレクトリを作成します。

● エラーコード… EACCES、ENOENT

● 例

次の例は、サブディレクトリsubdirを作成する例です。

```
#include <stdio.h>
#ifdef WIN32
#include <direct.h>
#else
#include <dir.h>
#endif

int main(void)
{
#ifdef WIN32
   if ( _mkdir("subdir") == 0)
#else
   if ( mkdir("subdir") == 0)
#endif
      puts("ディレクトリを作成しました。");
   else
      puts("ディレクトリを作成できませんでした。");

   return 0;
}
```

141 ディレクトリを作成する _mkdir()

ヘッダ	direct.h
書式	int _mkdir(const char **path*);
引数	*path*……作成するディレクトリのパス。
戻り値	新しいディレクトリが作成されたときは0、エラーのときは-1。

● 解説

ディレクトリを作成します。

● 参照→mkdir()

142 現地時刻をカレンダー値に変換する mktime()

ヘッダ	time.h
書式	time_t mktime(struct tm *timeptr);
引数	*timeptr* …変換する現地時刻を保存したtm構造体のポインタ。
戻り値	関数が正常に終了したときはtime_t型のカレンダー時刻値、そうでないときは(time_t)(-1)。

解説

tm構造体に入っている現地時刻をtime_t型の値に変換します。

参照→asctime()

143 浮動小数点実数を整数と小数部分に分ける modf()

ヘッダ	math.h
書式	double modf(double *x*, double **ptr*);
引数	*x* ………整数部と小数部に分ける実数。 *ptr* ……整数部を保存するdouble変数のポインタ。
戻り値	*x*の小数部分。

解説

浮動小数点値*x*を整数部と小数部に分割します。整数部と小数部の符号は同じになります。

参照→frexp()、ldexp()

144 プログラムが正常終了した時に呼び出す関数を登録する _onexit()

ヘッダ	stdlib.h
書式	_onexit_t _onexit(_onexit_t *func*);
引数	*func* ……呼び出す関数名。
戻り値	関数登録が成功すると関数のポインタ。そうでない場合はNULL。

解説

プログラムが正常終了した時に呼び出される関数を登録します。

参照→atexit()

145 ファイルやデバイスをオープンする open()

ヘッダ	sys/types.h、sys/stat.h、fcntl.h
書式	int open(const char **path*, int *flags*, [mode_t *mode*]);
引数	*path*……オープンするファイルやデバイスのパス。 *flags*……オープンするファイルに許可する操作を指定するフラグ。 *mode* …オープンするファイルのモード。省略可能です。
戻り値	新しいファイルデスクリプタ、エラーが発生したときは-1。

●解説

ファイルのオープンを試み、ファイルデスクリプタを返します。

●エラーコード

EACCES、EEXIST、EFAULT、EISDIR、ELOOP、EMFILE、ENAMETOOLONG、ENFILE、ENOENT、ENOMEM、ENOSPC、ENOTDIR、EROFS、ETXTBSY

●例

次の例は、test.txtという名前のファイルを作成して、文字列“Hello C/C++ world”を出力する例です。

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
#include <sys¥stat.h>

int main(void)
{
   int handle;
   char msg[] = "Hello C/C++ world";

#ifdef WIN32
   if ((handle = _open("test.txt",
       _O_CREAT | _O_TEXT, _S_IREAD | _S_IWRITE)) == -1)
#else
   if ((handle = open("test.tmp", O_CREAT | O_TEXT), S_IREAD|S_IWRITE) == -1)
#endif
   {
      perror("エラー");
      return 1;
   }
```

```
#ifdef WIN32
    _write(handle, msg, strlen(msg));
    _close(handle);
#else
    write(handle, msg, strlen(msg));
    close(handle);
#endif

    printf("%sを書き込みました。¥n", msg);

    return 0;
}
```

● 参照→creat()

146 ファイルやデバイスをオープンする _open()

ヘッダ	io.h
書式	int _open(const char **path*, int *flags* [, int *mode*]);
引数	*path*……オープンするファイルやデバイスのパス。 *flags*……オープンするファイルに許可する操作を指定するフラグ。 *mode* …オープンするファイルのモード。省略可能です。
戻り値	新しいファイルデスクリプタ、エラーが発生したときは-1。

● 解説

ファイルのオープンを試み、ファイルデスクリプタを返します。

● 参照→open()

プログラミング豆知識

コンパイラと標準

多くのコンパイラは、標準に準拠していることを謳っています。しかし、一方では、各コンパイラはそれぞれに言語や機能を拡張しています。しかも、多くの場合、デフォルトでは独自の拡張を優先しています。

ANSIの規格に完全に従うには、多くの場合、そのためのオプションを明示的に選択する必要があり、そのようなオプションを選択するとコンパイラ独自の拡張は使えなくなります。

147 ディレクトリを開く opendir()

ヘッダ	sys/types.h、dirent.h
書式	DIR *opendir(const char **name*);
引数	*name* …開くディレクトリ。
戻り値	成功したときはディレクトリストリームのポインタ、エラーが発生したときはNULL。

● 解説

指定した名前のディレクトリストリームを開き、そのポインタを返します。返されたポインタはディレクトリの先頭のエントリーを指していて、通常、これはreaddir()関数で使います。

● エラーコード

EACESS、EMFILE、ENFILE、ENOENT、ENOMEM、ENOTDIR

● 準拠… POSIX

● 例

次の例は、指定したディレクトリの内容を、最初は先頭の3個だけ表示し、次にすべての要素を表示します。

```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#ifdef __BORLANDC__
#pragma hdrstop
#include <condefs.h>
#include <conio.h>
#else
#include <sys/types.h>
#endif

int main(int argc, char *argv[])
{
   int i;
   char dirname[BUFSIZ];
   DIR *dir;
   struct dirent *ent;

   printf("ディレクトリ>");
```

```
    scanf("%s", dirname);

    if ((dir = opendir(dirname)) == NULL)
    {
        puts("ディレクトリをオープンできません。");
        exit(1);
    }

    printf("%s の最初の3個の内容:¥n", dirname);
    for (i=0; i<3; i++)
    {
        if ((ent = readdir(dir)) != NULL)
            printf("%s¥n",ent->d_name);
        else
            break;
    }

    /* ディレクトリの先頭に戻す */
    rewinddir(dir);
    printf("%s の内容:¥n", dirname);
    while ((ent = readdir(dir)) != NULL)
        printf("%s¥n",ent->d_name);
    closedir(dir);

    return 0;
}
```

148	シグナルを待つ pause()
ヘッダ	unistd.h
書式	int pause(void);
戻り値	常に-1。

●解説

シグナルを受け取るまで呼び出したプロセスをスリープさせます。この関数は常に-1を返し、errnoにエラーコードをセットします。

●互換性

これはPOSIXシステムのシステムコールとして実装されています。

●エラーコード… EINTER

149 パイプに結合されたファイルを閉じる pclose()

ヘッダ	stdio.h
書式	int pclose(FILE **fp*);
引数	*fp* ………ファイル（前に_popen()を呼び出したときの戻り値）。
戻り値	正常に終了しないときは-1。

● 解説

新しいコマンドプロセッサの完了まで待機し、パイプに結合されたファイルを閉じます。

● 互換性

Windowsでは`_pclose()`を使いstdio.hをインクルードします。

150 パイプに結合されたファイルを閉じる _pclose()

ヘッダ	stdio.h
書式	int _pclose(FILE **fp*);
引数	*fp* ………ファイル（前に_popen()を呼び出したときの戻り値）。
戻り値	正常に終了しないときは-1。

● 解説

新しいコマンドプロセッサの完了まで待機し、パイプに結合されたファイルを閉じます。

● 互換性… Windows

151 システムエラーメッセージを出力する perror()

ヘッダ	errno.h、stdio.h
書式	void perror(const char **s*);
引数	*s* ………表示する文字列メッセージ。

● 解説

文字列sとコロン（:)、およびシステムエラーメッセージを出力します。

● 参照→errno

152 パイプを作成する pipe()

ヘッダ	unistd.h
書式	int pipe(int *fd[2]*);
引数	*fd[2]*……ファイルデスクリプタを保存するための配列変数。
戻り値	成功したときは0、失敗したときは-1。

● 解説

読み書き用のパイプを作成します。引数はintの配列で、fd[0]には読み出し用、fd[1]には書き込み用のファイルデスクリプタが保存されます。

● エラーコード… EFAULT、EMFILE、ENFILE

153 パイプを作成する _pipe()

ヘッダ	io.h
書式	int _pipe(int **pfds*, unsigned int *psize*, int *mode*);
引数	*pfds* ……ファイルデスクリプタを保存する変数配列のポインタ。 *psize*……予約するメモリの容量。 *mode* …ファイルモード（_O_TEXTまたは_O_BINARY）。
戻り値	成功したときは0、失敗したときは-1。

● 解説

読み書き用のパイプを作成します。

● 互換性… Windows

● 参照→pipe()

154 パイプを作成してコマンドを実行する popen()

ヘッダ	stdio.h
書式	FILE *popen(const char **command*, const char **type*);
引数	*command* …実行するコマンド。 *type* ………返されるストリームの種類（読み出しは'r'、書き込みは'w'）。
戻り値	正常に終了できたときはファイルポインタ、そうでないときはNULL。

● 解説

パイプを作成してコマンドを実行します。

● 例

次の例は、パイプを作成して、ディレクトリの内容を表示するコマンドを実行するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>

int main( void )
{

   char   buffer[256];
   FILE   *pPipe;

#ifdef WIN32
   if( (pPipe = _popen( "dir *.* /on /p", "rt" )) == NULL )
#else
   if( (pPipe = popen( "ls -l", "r" )) == NULL )
#endif
      exit( 1 );

   /* パイプからEOFがあるまで読み込む */
   while(fgets(buffer, 256, pPipe))
   {
      printf(buffer);
   }

   if (feof( pPipe))
   {
#ifdef WIN32
     printf( "¥nプロセス戻り値=%d¥n", _pclose( pPipe ) );
#else
     printf( "¥nプロセス戻り値=%d¥n", pclose( pPipe ) );
#endif
   }
   else
   {
     printf( "エラー発生¥n");
   }
}
```

● 互換性… POSIX

155 パイプを作成してコマンドを実行する
_popen()

ヘッダ	stdio.h
書式	FILE *_popen(const char **command*, const char **type*);
引数	*command* …実行するコマンド。 *type* …………返されるストリームの種類（読み出しは'r'、書き込みは'w'）。
戻り値	正常に終了できたときはファイルポインタ、そうでないときはNULL。

● 解説

パイプを作成してコマンドを実行します。

● 参照→popen()

156 べキ乗を計算する
pow()

ヘッダ	math.h
書式	double pow(double *x*, double *y*);
引数	*x*、*y* ……*x*の*y*乗の値を計算するための値。
戻り値	*x*の*y*乗の値。

● 解説

*x*の*y*乗の値を計算します。

● エラーコード… EDOM

● 例

ldexp()の例参照。

157 書式付きデータを標準出力に書き込む
printf()

ヘッダ	stdio.h
書式	int printf(const char **fmt*[, argument, ...]);
引数	*fmt* ……出力するときの書式。解説参照。 *...* ………出力する値のリスト。
戻り値	成功したときは出力したバイト数、エラーの場合はEOF。

● 解説

書式化された出力を標準出力に書き出します。書式は次のような文字列で指定します。

%[flags] [width] [precision] [pre-type] type

各部分の詳細は以下のとおりです。

・flags

出力のフラグを指定します。

フラグ	意味
-	指定されたフィールド幅に結果を左詰めにする。
+	出力値が符号付きの場合は、出力値の前に符号（+または-）を付ける。
0	最小幅まで0が付加される。
空白	出力値が符号付きで整数の場合、出力値の前に空白を付ける。
#	書式がo、x、Xのとき0以外のすべての出力値の前に0、0x、0Xを付け、空白は出力しない。 書式がe、E、f、g、Gのとき、出力値に強制的に小数点を入れる。

・width

出力の文字幅（文字列にしたときの長さ）を指定します。

・precision

出力の小数点以下の文字幅を指定します。実数の場合であれば小数点以下の精度を指定します。

・pre-type

typeで指定する入力数値データ引数のデフォルトサイズを変更します。

プリタイプ	意味
h	データはshortまたはシングルバイト文字。
l、L	データはlongまたはワイド文字。I64のときは64ビット整数。
type	データの型を指定します。

タイプ	意味
c、C	1個の文字。
d	intを符号付き10進整数で出力。
i	intを符号付き8進整数で出力。
o	intを符号なし8進整数で出力。
p	ポインタを16進数で出力。
u	intを符号なし10進整数で出力。
x、X	intを符号なし16進整数で出力、"abcdef"を使用。Xのときは"ABCDEF"で出力。
e、E	doubleを[-]d.dddd e [+/-]ddd形式符号付きの値で出力。
f	doubleを[-]dddd.dddd形式の符号付きの値で出力。
g、G	doubleをfまたはeの書式のうち指定された値および精度を表現できる短い方の書式で出力。
s、S	文字列を最初のNULL文字('¥0')までまたはprecisionの範囲まで出力。Sの場合はワイド文字列。

出力文字列の書式にはさらに次のような決まりがあります。

・%そのものを出力するときは%%を使う
・¥、'、"などを出力するときにはその前に¥を付ける

●互換性

出力書式文字列は独自に拡張されている場合があります。

●例

次の例は、fgets()およびscanf()を使って文字列と整数および実数を入力し、printf()で書式を指定して出力する例です。

```
#include <stdio.h>
#define BUFFSIZE 256

int main(int argc, char *argv[])
{
    char buff[BUFFSIZE];
    int n;
    float f;
    double v;
    printf("文字列>");
    fgets(buff, BUFFSIZE, stdin);
```

```
    printf("整数実数>");
    scanf("%d %f", &n, &f);
    printf("文字列%s¥n", buff);
    v = (double)f;
    printf("%03d %x %5.2f %e¥n", n, n, f, v);

    return 0;
}
```

158 文字をファイルに書き込む putc()

ヘッダ	stdio.h
書式	int putc(int *c*, FILE * *fp*);
引数	*c* ………出力する文字。 *fp* ………出力するファイル。
戻り値	書き込まれた文字。エラーが発生したときはEOF。

● 解説

文字*c*を*fp*に出力します。マクロとして実装されている可能性があるという点を除いてfputc()と同じです。charはintにキャストして出力します。

159 文字を標準出力に書き込む putchar()

ヘッダ	stdio.h
書式	int putchar(int *c*);
引数	*c* ………出力する文字。
戻り値	書き込まれた文字。エラーが発生したときはEOF。

● 解説

文字*c*を標準出力に書き込みます。putc(c,stdout)と同じです。charはintにキャストして出力します。

160 環境変数を変更または追加する putenv()

ヘッダ	stdlib.h
書式	int putenv(const char **str*);
引数	*str* ………設定または追加する環境変数。
戻り値	成功したときは0、エラーが発生したときは-1。

● 解説

環境変数を追加するか値を変更します。この関数を使って変更した環境変数は、実行中のプロセスだけに影響を与えます。*str*には、たとえばLIB=/usr/libのような文字列を指定します。

● 参照→getenv()、setenv()

161 環境変数を変更または追加する _putenv()

ヘッダ	stdlib.h
書式	int _putenv(const char *str);

● 参照→putenv()

162 ストリームに1行書き込む puts()

ヘッダ	stdio.h
書式	int puts(const char **s*);
引数	*s* ………出力する文字列。
戻り値	成功したときは負でない数、エラーが発生したときはEOF。

● 解説

文字列sと改行を標準出力（stdout）に書き込みます。この関数を使って文字列を出力すると、必ず改行されます。出力文字列の最後に改行があると（たとえばabc ¥n）、改行が2回行われます。

163	配列を並べ替える qsort()
ヘッダ	stdlib.h
書式	void qsort(void **base*, size_t *n*, size_t *size*, int (**compare*)(const void *, const void *))
引数	*base* ………ソートする配列の先頭。 *n*……………配列の要素数。 *size* ………配列要素のサイズ（バイト数）。 *compare* …比較関数。

● **解説**

大きさ*size*の*n*個の要素をもつ配列*base*を、比較関数*compare*が返す値に従って並べ替えます。比較関数は、第1の引数が第2の引数に対して、小さいときは0未満、等しいときは0、大きいときは0より大きい整数を返さなければなりません。

● **例**… bsearch()、calloc()の例参照。

164	現在のプロセスにシグナルを送る raise()
ヘッダ	signal.h
書式	int raise (int *sig*);
引数	*sig*………送信するシグナル。
戻り値	成功したときは0、失敗したときは0以外の値。

● **解説**

現在のプロセスにシグナルを送ります。kill(getpid(),sig)と等価です。

*sig*には、次の値を指定します。

▼ 表　シグナルの値

シグナルの値	意味
SIGABRT	異常終了。
SIGFPE	不正な浮動小数点演算。
SIGILL	不当な命令。
SIGINT	Ctrl+Cによる割り込み。
SIGSEGV	記憶域に対する不正なアクセス。
SIGTERM	プログラム終了の要求。
SIGUSRn	ユーザー定義のシグナル（nは1、2、3などの数値）。
SIGBREAK	[Stop]（Ctrl+Break）による割り込み。

●例

次の例は、ゼロで割る計算が発生したときに、プログラムを終了するようにした例です。

```
#include <signal.h>

int main(void)
{
    int v1, v2;

    printf("値1>");
    scanf("%d", &v1);
    printf("値2>");
    scanf("%d", &v2);

    if (v2 == 0)
       raise(SIGFPE);

    printf("v1/v2=%d¥n", v1 / v2);

    return 0;
}
```

165 乱数を生成する rand()

ヘッダ	stdlib.h
書式	int rand(void);
戻り値	0～RAND_MAXの間の数。

●解説

0～RAND_MAXの間の疑似乱数整数を返します。この関数を使うときには、最初にsrand()を呼び出して、乱数ジェネレータを初期化してください。

●例

次の例は0～99までのランダムな整数を10個出力します。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
   int i;
   time_t t;

   srand((unsigned) time(&t));

   for (i=0; i<10; i++)
       printf("%d¥n", rand() % 100);

    return 0;
}
```

166 乱数を生成する random()

ヘッダ	stdlib.h
書式	long int random(void);
戻り値	0～RAND_MAXの間の値。

● 解説

乱数を生成して、0～RAND_MAXの範囲の疑似乱数を生成します。

● 準拠… BSD

● 互換性

この関数は、一部の処理系で実装されています。

● 参照→rand()、srand()

167 ファイルデスクリプタからデータを読み出す read()

ヘッダ	unistd.h
書式	ssize_t read(int *fd*, void **buf*, size_t *n*);
引数	*fd* ………データを読み出すファイルデスクリプタ。 *buf* ……読み込んだデータを保存するバッファのポインタ。 *n* ………読み出す最大バイト数。
戻り値	成功したときは読み込んだバイト数、ファイルの終わりに達したら0、エラーが発生したときは-1

解説

ファイルデスクリプタ*fd*から最大*n*バイトを*buf*で始まるバッファへ読み込みます。

エラーコード

EAGAIN、EBADF、EFAULT、EINTR、EINVAL、EIO、EISDIR

例

次の例は、read()を使ってファイルから3バイト取得する例です。

```
#include <stdio.h>
#ifdef WIN32
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>
#include <stdio.h>
#else
#include <unistd.h>
#endif

int main(int argc, char *argv[])
{
   char buf[128];
   int fh, n;
#ifdef WIN32
   if((fh = _open("./test.dat", _O_RDONLY)) == -1)
#else
   if((fh = open("./test.dat", O_RDONLY)) == -1)
#endif
   {
```

```
        printf("ファイルを開けません.¥n");
        return -1;
    }
#ifdef WIN32
    n = _read(fh, &buf, 3);
    _close(fh);
#else
    n = read(fh, &buf, 3);
    close(fh);
#endif
    if (n != 0) {
        buf[n] = 0;
        printf("%s¥n", buf);
    }
    return 0;
}
```

168 ファイルデスクリプタからデータを読み出す _read()

ヘッダ	io.h
書式	int _read(int *fd*, void **buf*, unsigned int *count*);
引数	*fd* ………データを読み出すファイルデスクリプタ。 *buf* ……読み込んだデータを保存するバッファのポインタ。 *count* …読み出す最大バイト数。
戻り値	成功したときは読み込んだバイト数、ファイルの終わりに達したら0、エラーが発生したときは-1

● **解説**

ファイルデスクリプタ*fd*から最大*count*バイトを*buf*で始まるバッファへ読み込みます。

● **準拠**… ISO C++

● **参照**→read()

169	ディレクトリを読み込む readdir()
ヘッダ	sys/types.h、dirent.h
書式	struct dirent *readdir(DIR *dir);
引数	*dir*………読み込むディレクトリを表すディレクトリストリームデスクリプタ（通常はopendir()が返した値）。
戻り値	dirent構造体のポインタ、ファイルの最後に達した場合やエラーが発生した場合はNULL。

●解説

*dir*に連結しているディレクトリストリームの情報を含むdirent構造体のポインタを返します。

●エラーコード… EBADF

●準拠… POSIX

●例

opendir()の例参照。

●注意

この関数はPOSIX関数ですが、dirent構造体にはサイズが明示されていないフィールドが含まれています。Win32ではNFSクライアントがRPC（Remote Procedure Call、リモートプロシージャコール）を実行するときにこの関数を使うことができます。

170	メモリを動的に再割り当てする realloc()
ヘッダ	stdlib.h
書式	void *realloc(void *pmem, size_t size);
引数	*pmem* …すでに割り当てられているメモリブロックへのポインタ。 *size* ……新しいサイズ（バイト単位）。
戻り値	成功したときは新たに割り当てられたメモリのポインタ、そうでないときにはNULL。

●解説

ポインタ*pmem*が示すメモリブロックのサイズを変更して*size*バイトにします。メモリに含まれる内容は変更されず、新しく割り当てられたメモリの内容も初期化されません。*pmem*にNULLを指定した場合には、malloc(size)と同じです。

●例

次の例は、メモリを動的に確保するプログラムの例です。

```
#include <memory.h>
#include <string.h>
#include <stdio.h>
#ifdef WIN32
#include <malloc.h>
#endif

int main(int argc, char *argv[])
{
   char *ps;

   /* メモリを割り当てて0で初期化する */
   ps = (char *)malloc(128);
   memset(ps, 'a', 128);
   ps[127] = 0;

   /* サイズを表示する */
   printf("psの長さ=%d¥n", strlen(ps));

   /* メモリを再割り当てする */
   ps = (char *)realloc(ps, 256);
   ps[127] = 'a';
   ps[255] = 0;

   /* サイズを表示する */
   printf("psの長さ=%d¥n", strlen(ps));

   /* メモリを解放する */
   free(ps);

   return 0;
}
```

171 ソケットからメッセージを受け取る recv()

ヘッダ	sys/types.h、sys/socket.h
書式	int recv(int *s*, void **buf*, int *len*, unsigned int *flags*);
引数	*s* ………ソケットを識別するデスクリプタ。 *buf* ……受信データを受け取るバッファ。 *len* ……バッファbufの長さ。 *flags*……フラグ（解説参照）。
戻り値	成功したときは受信したバイト数、そうでないときは-1。

● 解説

接続されたソケット*s*からデータを受信します。

*flags*には以下の値をOR(|)で結合して指定します。

▼表　flagsの値

flagsの値	意味
MSG_OOB	通常のデータストリームで受信できない帯域外データを処理する。
MSG_PEEK	入って来たメッセージを調べる。
MSG_WAITALL	要求を完全に満たすかエラーが発生するまで待つ。

172 ファイルを削除する remove()

ヘッダ	stdio.h、io.h
書式	int remove(const char **path*);
引数	*path*……削除するファイルのパス名。
戻り値	成功したときは0、失敗したときは-1。

● 解説

*path*で指定されたファイルを削除します。ファイルを削除する前にそのファイルのハンドルをすべて閉じる必要があります。

● エラーコード… EACCES、ENOENT

例

```
#include <stdio.h>

int main(void)
{
    char fname[256];

    /* 削除するファイル名を入力する */
    printf("削除するファイル名>");
    gets(fname);

    /* ファイルを削除する */
    if (remove(fname) == 0)
       printf("%s は削除されました。¥n", fname);
    else
       printf("%s を削除できません。¥n", fname);

    return 0;
}
```

参照→_unlink

173 ファイル名やディレクトリ名を変更する rename()

ヘッダ	stdio.h
書式	int rename(const char **oldpath*, const char **newpath*);
引数	*oldpath* ………変更前のファイルのパス。 *newpath*………変更後のファイルのパス。
戻り値	成功したときは0、エラーが発生したときは-1。

解説

ファイルやディレクトリの名前を変更します。その際に必要ならば別のディレクトリに移動します。

エラーコード

EACCES、EBUSY、EEXIST、EFAULT、EINVAL、EISDIR、ELOOP、EMLINK、ENAMETOOLONG、ENOENT、ENOMEM、ENOSPC、ENOTEMPTY、ENOTDIR、EPERM、EROFS、EXDEV

174 ファイルポインタの位置をファイルの先頭に移動する rewind()

ヘッダ	stdio.h
書式	void rewind(FILE **fp*);
引数	*fp* ………ポインタを移動するファイル。

● 解説

ファイル*fp*の中の位置を示すポインタをファイルの先頭に移動します。(void)fseek(stream, 0L, SEEK_SET)と同じですが、rewind()を呼び出したときにはストリームに対するエラーインジケータもクリアされます。

● **エラーコード**… EBADF、EINVAL、SEEK_END

175 書式付きデータを標準入力から読み出す scanf()

ヘッダ	stdio.h
書式	int scanf(const char **fmt*, ...);
引数	*fmt* ……読み出すデータの書式。 ... ………読み込んだデータを保存するための変数のポインタ。

● 解説

指定された書式で標準入力からデータを読み出します。データは、空白、タブ、改行で区切られているものと解釈されます。

書式は次のような文字列で指定します。

%[*] [width] [pre-type]type

各部分の詳細は以下のとおりです。

・*

入力の値を引数に保存しない。

・width

入力の文字幅（文字列にしたときの長さ）を指定します。

・pre-type

typeで指定する入力数値データ引数のデフォルトサイズを変更します（printf()参照）。

・type

データの型を指定します。

タイプ	意味
h	データはshortまたはシングルバイト文字。
l、L	データはlongまたはワイド文字。I64のときは64ビット整数。
c、C	1個の文字。
d	intを符号付き10進整数で入力。
i	intを符号付き8進整数で入力。
o	intを符号なし8進整数で入力。
u	intを符号なし10進整数で入力。
x、X	intを符号なし16進整数、"abcdef"を使用。Xのときは"ABCDEF"で入力。
e、E	floatを[-]d.dddd e [+/-]ddd形式符号付きの値で入力。
f	floatを[-]dddd.dddd形式の符号付きの値で入力。
g、G	floatをfまたはeの書式のうち指定された値および精度を表現できる短い方の書式で入力。
s、S	文字列を最初のNULL文字('¥0')までまたはprecisionの範囲まで入力。Sの場合はワイド文字列。

● **注意**

通常、printf()系のデフォルトの実数の型はdoubleですが、scanf()系の実数の型はfloatです。実数はfloat変数で受け取ってからdoubleに変換する必要があります。

● **例**… printf()の例参照。

176 ソケットへメッセージを送る send()

ヘッダ	sys/types.h、sys/socket.h
書式	int send(int *s*, const void **msg*, int *len*, unsigned int *flags*);
引数	*s* ………ソケットを識別するデスクリプタ。 *msg* ……送信するメッセージのポインタ。 *len* ……メッセージの長さ。 *flags*……フラグ。
戻り値	送ったキャラクタ数。エラーが発生したときは-1。

● **解説**

ソケット*s*にメッセージを転送します。

*flags*には以下の値をOR(|)で結合して指定します。

▼表　flagsの値

flagsの値	意味
MSG_OOB	通常のデータストリームで送信できない帯域外データを処理する。
MSG_DONTROUTE	ルーティングをバイパスして直接送る。
エラーコード	EBADF、EFAULT、EMSGSIZE、ENOBUFS、ENOTSOCK、EWOULDBLOCK

177 ファイルのバッファリングを制御する setbuf()

ヘッダ	stdio.h
書式	void setbuf(FILE *fp, char *buf);
引数	*fp* ………ファイル。 *buf* ……ストリームに割り当てるバッファのポインタ。

● **解説**… ファイルにバッファを割り当てます。

● **注意**… 新しいプログラムではsetbuf()ではなくsetvbuf()を使ってください。

178 環境変数を設定する setenv()

ヘッダ	stdlib.h
書式	int setenv(const char *name, const char *value, int overwrite);
引数	*name* ………環境変数の名前。 *value* ………環境変数に設定する値。 *overwrite* …nameがすでに環境に存在する場合の変更の方法(解説参照)。
戻り値	成功したときは0、そうでないときは-1。

● **解説**

*name*が存在しないときは環境変数*name*に値*value*を設定して環境に追加します。*overwrite*には以下のいずれかで指定します。

▼表　overwriteの値

overwriteの値	意味
0	nameの値は変更しない。
0以外	nameの値をvalueに変更する。

*name*が存在するときは、*overwrite*が0以外ならばその値を*value*に変更し、*overwrite*が0ならば*name*の値は変更しません。

●例

次の例は、環境変数を変更するプログラムの例です。

```
#include <stdlib.h>
#include <stdio.h>

#define BUFFSIZE 2048

int main(int argc, char *argv[])
{
   char envbuf[BUFFSIZE], envname[BUFFSIZE];
   char buff[BUFFSIZE], envstr[BUFFSIZE], *envvar;
   int envIsExist = 1;

   printf("調べる環境変数名>");
   scanf("%s", envname);

   /* 現在の環境変数の値を取得する */
   envvar = getenv(envname);
   if (envvar != NULL)
   {
      strcpy(envbuf, envvar);
      printf("環境変数%s の内容: %s¥n", envname, envbuf);
   }
   else
      envIsExist = 0;

   /* 環境変数を変更する */
   printf("新しい環境変数の値>");
   scanf("%s", buff);
#ifdef WIN32
   sprintf(envstr, "%s=%s", envname, buff);
   if (_putenv(envstr))
#else
   if (setenv(envname, buff, 1))
#endif
      printf("環境変数の設定に失敗しました.¥n");

   /* 新しい環境変数の値を取得する */
   envvar = getenv(envname);
   if(envvar != NULL)
```

```
    printf("環境変数%s の内容: %s¥n", envname, envvar);
    if (envIsExist)
    {
        /* 環境変数をもとに戻す */
#ifdef WIN32
        sprintf(envstr, "%s=%s", envname, envbuf);
        if (_putenv(envstr) == -1)
#else
        if (setenv(envname, envbuf, 1)== -1)
#endif
            printf("環境変数の設定に失敗しました.¥n");
    }

    return 0;
}
```

● **参照**→getenv()、putenv()

179 プログラムの現在の環境を保存する setjmp()

ヘッダ	setjmp.h
書式	int setjmp(jmp_buf *env*);
引数	*env* ……環境を保存するjmp_buf構造体変数。
戻り値	スタック環境の保存に成功したときは0、関数longjmp()を呼び出した結果としてこの関数がリターンするかのような動作をするときはlongjmp()の引数val（ただし、valが0の場合は1）。

● **解説**

longjmp()で使うjmp_buf構造体変数*env*にスタック環境を保存します。

● **互換性**

関数setjmp()が保存する環境値や関数longjmp()が復元する環境値は、実行環境によって異なります。

● **参照**→longjmp()

180 現在のロケールを設定する setlocale()

ヘッダ	locale.h
書式	char *setlocale(int *category*, const char * *locale*);
引数	*category*……設定したロケールの影響を受けるカテゴリ。 *locale* ………ロケール名。"C"、"POSIX"、"Japanese"など。
戻り値	成功したときはロケールセットに対応する文字列、そうでないときはNULL。

● 解説

プログラムの現在のロケールを設定します。ロケールはキャラクタや文字列の評価、数字、日付、貨幣の表示形式などに影響を与えます。実際に与える影響の詳細は環境によって異なります。指定可能な値は環境によって異なることがあるので、戻り値をチェックすることを推奨します。

*category*には以下のいずれかを指定します。

▼表 categoryの値

categoryの値	意味
LC_ALL	すべてに影響を与える。
LC_COLLATE	関数strcoll()とstrxfrm()に影響を与える。
LC_CTYPE	文字種別判定と文字変換を行うルーチンに影響を与える。
LC_MONETARY	localeconv()に影響を与える。
LC_NUMERIC	10進数字に影響を与える。
LC_TIME	strftime()に影響を与える。

この関数のワイド文字バージョンは_wsetlocale()です。

● 例

isalnum()の例参照。

181 ファイルシステム記述ファイルをオープンする setmntent()

ヘッダ	stdio.h、mntent.h
書式	FILE *setmntent(const char **fname*, const char **type*);
引数	*fname* …オープンするファイルシステム記述ファイルのファイル名。 *type* ……ファイルのオープンモード（解説参照）。
戻り値	成功したときはオープンしたファイルのポインタ。そうでないときはNULL。

● **解説**

ファイルシステム記述ファイルをオープンし、getmntent()を呼び出すときに使うファイルポインタを返します。*type*にはfopen()のmode引数と同じ値を指定できます。

mntent構造体は次のように定義されています。

```
struct mntent {
    char *mnt_fsname; /* マウントされているファイルシステムの名前 */
    char *mnt_dir; /* ファイルシステムのパスプリフィックス */
    char *mnt_type; /* マウントタイプ（mntent.h参照） */
    char *mnt_opts; /* マウントオプション（mntent.h参照） */
    int mnt_freq; /* dumpでダンプする必要があるかどうかを示す */
    int mnt_passno; /* ブート時にファイルチェックの必要があるかどうかを示す */
};
```

● **仕様**… BSD 4.3

● **例**… getmntent()の例参照。

● **関連ファイル**

/etc/fstab ……ファイルシステム記述ファイル

/etc/mtab ……マウントされたファイルシステムの記述ファイル

● **参照**→fopen()

182　ファイルのバッファリングとバッファのサイズを制御する
setvbuf()

ヘッダ	stdio.h
書式	int setvbuf(FILE **fp*, char **buf*, int *type*, size_t *size*);
引数	*fp* ………バッファリングを設定するファイル。 *buf* ……バッファのポインタ。 *type* ……バッファリングの種類（解説参照）。 *size* ……バッファの大きさ。
戻り値	成功したときは0、エラーが発生したときは0以外の値。

● **解説**

ファイルにバッファを割り当てます。ファイルのバッファは、通常、自動的に割り当て

られますが、この関数を使ってバッファを割り当てると、入出力バッファとして割り当てたバッファ*buf*をオープンして使えるようになります。

バッファの大きさは、0より大きく、limits.hで定義されている定数UINT_MAX以下の値でなければなりません。

type（バッファリングの種類）は以下のいずれかです。

▼表 typeの値

typeの値	意味
_IOFBF	ファイルは完全にバッファリングされる。バッファに空きがあると、入力動作はバッファがいっぱいになるまで行われる。出力時には、バッファが完全にいっぱいにならなければ、データはファイルに書き込まれない。
_IOLBF	ファイルは行バッファリングされる。バッファに空きがあると、入力動作はバッファがいっぱいになるまで行われる。出力時には、改行文字がファイルに書き込まれると、バッファをファイルにフラッシュする。
_IONBF	ファイルはバッファリングされない。引数bufとsizeは無視される。各入力動作はファイルから直接読み出し、各出力動作はただちにデータをファイルに書き込む。

183 割り込みシグナル処理を設定する signal()

ヘッダ	signal.h
書式	void (__cdecl *signal(int *sig*, void (__cdecl **func*) (int [, int]))) (int);
引数	*sig*………シグナル値。 *func* ……シグナルハンドラ。
戻り値	以前のシグナルハンドラの値。エラーの場合はSIG_ERR。

●解説

オペレーティングシステムからの割り込みシグナルに対処する方法やそれを処理する関数（シグナルハンドラ）を設定します。シグナルハンドラ*func*の最初のパラメータはシグナル値、2番目のパラメータは最初のパラメータがSIGFPEの場合に指定できるサブコードです。

●互換性

この関数の動作や引数に指定可能な値は実行環境に依存します。

184	サインを計算する sin()
ヘッダ	math.h
書式	double sin(double *x*);
引数	*x* ………サインの値を計算したい角度（ラジアン単位）。
戻り値	成功したときはサインの値（-1～1）、そうでない場合は戻り値は不定。

解説

正弦（サイン）の値を返します。関数が失敗した場合は、errnoにエラーコードがセットされます。

例

次の例は、ユーザーが入力した角度に対する、サイン、コサイン、タンジェント、アークサイン、アークコサイン、アークタンジェントをそれぞれ出力するプログラムの例です。

```
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[])
{
   float r;
   printf("角度（ラジアン）>");
   scanf("%f", &r);

   printf("サイン= %.3f¥n", sin(r));
   printf("コサイン= %.3f¥n", cos(r));
   printf("タンジェント= %.3f¥n", tan(r));
   printf("アークサイン= %.3f¥n", asin(r));
   printf("アークコサイン= %.3f¥n", acos(r));
   printf("アークタンジェント= %.3f¥n", atan(r));

   return 0;
}
```

185	ハイパーボリックサインを計算する sinh()
ヘッダ	math.h
書式	double sinh(double *x*);
引数	*x* ………ハイパーボリックサインの値を計算したい角度（ラジアン単位）。

解説

ハイパーボリックサイン（双曲線正弦）関数の値を返します。

これは、(exp(x) - exp(-x)) / 2で計算される値です。

186	通信ソケットを作成する socket()
ヘッダ	sys/types.h、sys/socket.h
書式	int socket(int *domain*, int *type*, int *protocol*);
引数	*domain* ………通信を行なうドメイン。 *type* …………ソケットの種類（解説参照）。 *protocol* ……プロトコルを指定する定数値。
戻り値	ソケットの作成に成功したときはソケットを識別するデスクリプタ、エラーが発生したときは-1。

● 解説

通信ソケットを作成してデスクリプタを返します。*domain*には以下のいずれかを指定します。

▼表　domainの値

domainの値	意味
PF_UNIX,PF_LOCAL	ローカル通信。
PF_INET	IPv4インターネット・プロトコル。
PF_INET6	IPv6インターネット・プロトコル。
PF_IPX	IPX-Novellプロトコル。
PF_NETLINK	カーネル・ユーザ・デバイス。
PF_X25	ITU-TX.25/ISO-8208プロトコル。
PF_AX25	アマチュア無線AX.25プロトコル。
PF_ATMPVC	生のATMPVCにアクセスする。
PF_APPLETALK	アップルトーク。
PF_PACKET	下層のパケットインターフェース。

*type*には以下の値を組み合わせた値を指定します。

▼表　typeの値

typeの値	意味
SOCK_STREAM	順序性と信頼性がある双方向の接続されたバイトストリーム。
SOCK_DGRAM	データグラム（信頼性無し、固定最大長メッセージ）。
SOCK_SEQPACKET	データグラム（信頼性有り、順序保障、固定最大長）。
SOCK_RAW	生のネットワークプロトコル。
SOCK_RDM	データグラム（信頼性有り、順序は保証しない）。
SOCK_PACKET	以前との互換性のため。新しいプログラムでは使わない。

● **エラーコード**… EACCES、EMFILE、ENFILE、ENOBUFS、EPROTONOSUPPORT

187 書式付きデータを文字列に書き込む sprintf()

ヘッダ	stdio.h
書式	int sprintf(char **str*, const char **fmt*, ...);
引数	*str* ………結果を書き込む文字列のポインタ。 *fmt* ……データを書き込むときの書式を指定する文字列のポインタ。printf()参照。 ... ………出力する任意の数の値を順に記述します。

● 解説

書式*fmt*に従ってデータを文字列バッファ*str*に出力します。printf()は標準出力に出力しますが、この関数は代わりに文字列に書き込むこと以外、printf()と同じです。

● **参照**→printf()

188 平方根を計算する sqrt()

ヘッダ	math.h
書式	double sqrt(double *x*);
引数	*x* ………平方根を求める値。正の値でなければなりません。
戻り値	平方根の値。

● 解説

*x*の平方根のうち、負でない値を返します。

● **エラーコード**… EDOM（xが負のとき）

189 乱数系列を初期化する srand()

ヘッダ	stdlib.h
書式	void srand(unsigned int *seed*);
引数	*seed* ……乱数を生成するときのシード値。

● 解説

関数rand()で生成する疑似乱数の整数系列の新しいシード値を設定します。

● **参照**→rand()

190	乱数系列を初期化する **srandom()**
ヘッダ	stdlib.h
書式	void srandom(unsigned int *seed*);
引数	*seed* ……乱数を生成するときのシード値。

● 解説

関数random()で生成する疑似乱数の整数系列の新しいシード値を設定します。

● 参照→random()

191	書式付きデータを文字列から読み出す **sscanf()**
ヘッダ	stdio.h
書式	int sscanf(const char **str*, const char **fmt*, ...);
引数	*str* ………データを読み出す文字列のポインタ。 *fmt* ……読み出すデータの書式。scanf()参照。 *...* ………読み込んだデータを保存するための変数のポインタ。
戻り値	読み出されたデータの数。

● 解説

文字列*str*から書式*fmt*に従ってデータを読み出します。

● 参照→scanf()

192	ファイルについての情報を取得する **stat()**
ヘッダ	sys/stat.h、unistd.h
書式	int stat(const char **path*, struct stat **buf*);
引数	*path* ……情報を取得するファイルのパス。 *buf* ……取得した情報を保存するstat構造体変数のポインタ。
戻り値	成功したときは0、エラーが発生したときは-1。

● 解説

ファイルまたはディレクトリに関する情報を取得します。

● 参照→fstat()

193 ファイルについての情報を取得する
_stat()

ヘッダ	sys/types.hとsys/stat.h
書式	int _stat(const char **path*, struct _stat **buf*);
引数	*path* ……情報を取得するファイルのパス。 *buf* ……取得した情報を保存する_stat構造体変数のポインタ。
戻り値	成功したときは0、エラーが発生したときは-1。

● 解説

ファイルまたはディレクトリに関する情報を取得します。

_stat構造体はWindowsの32ビットでは次のように定義されています。

```
struct _stat32i64 {
        _dev_t      st_dev;
        _ino_t      st_ino;
        unsigned short st_mode;
        short       st_nlink;
        short       st_uid;
        short       st_gid;
        _dev_t      st_rdev;
        __int64     st_size;
        __time32_t st_atime;
        __time32_t st_mtime;
        __time32_t st_ctime;
        };
```

● 参照→stat()

194 2つの文字列を連結する
strcat()

ヘッダ	string.h
書式	char *strcat(char **dest*, const char **src*);
引数	*dest* ……先頭の文字列であり、連結した結果を保存することになる文字列バッファ。 *src*………後ろに連結する文字列。
戻り値	結果としてできた文字列destのポインタ。

● 解説

文字列*dest*のあとに文字列*src*を付け加えます。*dest*の最後にある¥0文字は*src*の最初の文字で上書きされます。

195 文字列中の文字の位置を示す strchr()

ヘッダ	string.h
書式	char *strchr(const char **s*, int *c*);
引数	*s* ………文字を探す文字列。 *c* ………探す文字。
戻り値	成功したときは一致した最初の文字のポインタ、文字が見つからないときはNULL。

● 解説

文字列*s*の中に文字*c*があるかどうか調べます。この文字*c*は1バイトの文字のことであり、マルチバイト文字やワイド文字にはこの関数は使えません。

196 2つの文字列を比較する strcmp()

ヘッダ	string.h
書式	int strcmp(const char **s1*, const char **s2*);
引数	*s1*、*s2* …比較する文字列。
戻り値	比較した結果を示す整数（解説参照）。

● 解説

2つの文字列*s1*と*s2*を比較します。比較は文字コードの値で行われ、最初に一致しない文字コード値の大小で関係が決定されます。

戻り値の意味は次の表に示すとおりです。

▼ 表 strcmp()の戻り値の意味

返される値	文字列s1とs2の関係
0未満の値	s1がs2より小さい。
0	s1とs2が等しい。
0を超える値	s1がs2より大きい。

● 参照→strcoll()、strncmp()

● 例

次の例は、入力された2つの文字列を比較してその結果を出力する例です。

```
#include <stdio.h>

int main(void)
{
   int ret;
   char s1[256], s2[256];

    /* 文字列を入力する */
    printf("文字列1>");
    gets(s1);
    printf("文字列2>");
    gets(s2);

    /* 文字列を比較する */
    ret = strcmp(s1, s2);
    if (ret == 0)
       printf("%s と %s は同じ。¥n", s1, s2);
    else if (ret > 0)
       printf("%s より %s が小さい。¥n", s1, s2);
    else
       printf("%s より %s が大きい。¥n", s1, s2);

    return 0;
}
```

197 現在のロケールを使って2つの文字列を比較する strcoll()

ヘッダ	string.h
書式	int strcoll(const char **s1*, const char **s2*);
引数	*s1*、*s2* …比較する文字列。
戻り値	比較した結果を示す整数（解説参照）。

● 解説

2つの文字列*s1*と*s2*を、カテゴリLC_COLLATEに対するプログラムの現在のロケールに適切であると解釈された文字列に基づいて比較します。ロケールがPOSIXまたはCの場合、strcoll()はstrcmp()と同じです。

戻り値の意味は次の表に示すとおりです。

▼表　strcoll()の戻り値の意味

返される値	文字列s1とs2の関係
0未満の値	s1がs2より小さい。
0	s1とs2が等しい。
0を超える値	s1がs2より大きい。

● 参照→strcmp()

198 文字列をコピーする strcpy()

ヘッダ	string.h
書式	char *strcpy(char **dest*, const char **src*);
引数	*dest* ……コピーした結果を保存する文字列バッファのポインタ。 *src*………コピー元の文字列のポインタ。
戻り値	コピーされた文字列destのポインタ。

● 解説

文字列*src*をバッファ*dest*にコピーします。*src*の最後はNULLで終了していなければならず、終端のNULL（¥0）もコピーされます。

199 文字列から特定の文字セットを含まない部分を探す strcspn()

ヘッダ	string.h
書式	size_t strcspn(const char **str1*, const char **str2*);
引数	*str1*……*str2*の一連の文字のいずれかが含まれているかどうかを調べる文字列。 *str2*……含まれない文字を保存したNULLで終わるキャラクタ配列。
戻り値	文字列*str2*のいずれかの文字以外の文字が*str1*に最初に現れたときの*str1*のその位置までの*str1*の文字列の長さ。

● 解説

*str2*で指定した一連の文字のいずれかが*str1*に含まれているかどうかを調べます。

● 互換性

この関数のワイド文字バージョンはwcscspn()、マルチバイトバージョンは_mbscspn(const unsigned char *string1, const unsigned char *string2)です。

200 エラーコードを説明する文字列を取得する strerror()

ヘッダ	string.h
書式	char *strerror(int *errnum*);
引数	*errnum* …エラーコード。
戻り値	エラーコードを説明する文字列。

● 解説

引数errnumに対応するエラーコードを説明する文字列を返します。エラーコードが不明な場合には不明であることを表すエラーメッセージ文字列が返されます。

201 日付と時間を書式化する strftime()

ヘッダ	time.h
書式	size_t strftime(char **s*, size_t *max*, const char **fmt*, const struct tm **t*);
引数	*s* ………書式化した文字列を保存するバッファのポインタ。 *max* ……文字列の最大の長さ。 *fmt* ……書式制御文字列。 *t* ………tデータ構造体。
戻り値	成功したときはバッファに保存した文字列の文字数、エラーが発生したときは0。

解説

tmデータ構造体のポインタptmが指す時刻を書式*fmt*に従って書式化し、文字列にして*str*に保存します。時刻の値は、現在のロケールのカテゴリLC_TIMEに従って書式化されます。

ANSI形式の書式指定子は次のとおりです。

▼表 ANSI形式の書式指定子

書式指定子	変換
%%	文字%。
%#%	文字%（フラグは無視される）。
%a	曜日の省略形。
%#a	曜日の省略形（フラグは無視される）。
%A	正式な曜日名。
%#A	正式な曜日名（フラグは無視される）。
%b	月の省略形。
%#b	月の省略形（フラグは無視される）。
%B	正式な月名。
%#B	正式な月名（フラグは無視される）。
%c	日付と時刻。
%#c	現在の地域情報に基づく長い形式の日付と時刻（例:Monday,April10,2000,10:11:22)。
%d	常に2桁で表す日付（01～31）。
#d	日付。
%H	常に2桁で表す24時間表記の時（00～23）。
%#H	24時間表記の時。
%I	常に2桁で表す12時間表記の時（01～12）。

03-03　関数の説明

%#I	12時間表記の時。
%j	常に3桁で表すその年の通し日数（001～366）。
%#j	その年の通し日数。
%m	常に2桁で表す月（01～12）。
%#m	10進数の月。
%M	常に2桁で表す分（00～59）。
%#M	分。
%p	AMまたはPM。
%#p	AMまたはPM（フラグは無視される）。
%S	常に2桁で表す秒（00～59）。
%#S	秒。
%U	日曜を週の1日目として年初から数えた常に2桁で表す週番号（00～53）。
%#U	日曜を週の1日目として年初から数えた週番号。
%w	0を日曜日とした曜日（0～6）。
%#w	0を日曜日とした曜日。
%W	月曜を週の1日目として年初から数えた常に2桁で表す週番号（00～53）。
%#W	月曜を週の1日目として年初から数えた週番号。
%x	日付。
%#x	現在の地域情報に基づく長い形式の日付。
%X	時刻。
%#X	時刻（フラグは無視される）。
%y	世紀を表す最初の2桁を除いた常に2桁で表す西暦年（00～99）。
%#y	世紀を表す最初の2桁を除いた西暦年。
%Y	4桁の西暦年。
%#Y	4桁の西暦年（先頭部分の0があれば取り除く）。
%Z,%z	時間帯名（時間帯がなければ文字は入らない）。
%#Z	時間帯名（時間帯がなければ文字は入らない、フラグは無視される）。

POSIX 形式の書式指定子は次のとおりです。

▼表 POSIX形式の書式指定子

書式指定子	変換
%C	10進数で表す世紀（00～99）
%D	mm/dd/yyの書式の日付
%e	月の日付（1～31）
%h	%bと同じ
%n	改行文字
%r	am/pm文字列が付いた12時間（01～12）の時刻。%I:%M:%S %pと同じ
%t	タブ文字
%T	HH:MM:SSの書式の24時間（00～23）の時刻
%u	10進数で表す曜日（月曜日は1、日曜日は7）

●注意

ANSIとPOSIXの両方をサポートする場合、環境によってはPOSIXで定義された指定子を使うために、__USELOCALES__を定義しなければなりません。また、実際にサポートされている書式指定子がこれとは異なる場合があります。

●例

次の例は、現在の日時をローカル時刻で表示するプログラムの例です。

```
#include <stdio.h>
#include <time.h>

int main(void)
{
   struct tm *ptm;
   time_t now;
   char str[80];

#ifdef WIN32
   _tzset();
#else
   tzset();
#endif
   time(&now);
   ptm = localtime(&now);
   strftime(str, 80, "現在の(%Z)=19%y/%A/%B(%d)%I:%M", ptm);
   puts(str);

   return 0;
}
```

202	文字列の長さを取得する strlen()
ヘッダ	string.h
書式	size_t strlen(const char *str);
引数	*str*………長さを求める文字列。
戻り値	文字列の中の文字数（バイト数）。

● **解説**

文字列*str*の長さを計算します。長さは文字列の最後にあるNULL（¥0）までで、NULLそのものは計算に含まれません。

● **例**… creat()、_dup()、fgets()、fwrite()などの例参照。

203	2つの文字列を連結する strncat()
ヘッダ	string.h
書式	char *strncat(char *dest, const char *src, size_t n);
引数	*dest* ……先頭の文字列であり、連結した結果を保存することになる文字列バッファ。 *src*………後ろに連結する文字列。 *n* ………連結する文字の数。
戻り値	結果としてできる文字列*dest*のポインタ。

● **解説**

文字列*dest*のあとに文字列*src*を*n*文字だけ付け加えます。*dest*の最後にある¥0文字は*src*の最初の文字で上書きされます。

● **参照**→strcat()

204	2つの文字列を比較する strncmp()
ヘッダ	string.h
書式	int strncmp(const char *s1, const char *s2, size_t n);
引数	*s1*、*s2* …比較する文字列。 *n* ………比較する文字数。
戻り値	比較した結果を示す整数（解説参照）。

● 解説

2つの文字列*s1*と*s2*の先頭から*n*バイトを比較します。比較は文字コードの値で行われ、最初に一致しない文字コード値の大小で関係が決定されます。

戻り値の意味は次の表に示すとおりです。

▼表　strncmp()の戻り値の意味

返される値	文字列s1とs2の関係
0未満の値	s1がs2より小さい。
0	s1とs2が等しい。
0を超える値	s1がs2より大きい。

● 参照→strcmp

205 文字列をコピーする strncpy()

ヘッダ	string.h
書式	char *strncpy(char **dest*, const char **src*, size_t *n*);
引数	*dest* ……コピーした結果を保存する文字列バッファのポインタ。 *src*………コピー元の文字列のポインタ。 *n* ………コピーする文字数。
戻り値	コピーされた文字列*dest*のポインタ。

● 解説

文字列*src*の先頭から*n*バイトをバッファ*dest*にコピーします。*n*が*src*より長い場合には、*dest*には*n*の長さまでNULL文字（¥0）が付けられます。

● 注意

コピーされた文字列の最後にNULL文字（¥0）が必ず付加されるわけではありません。

● 参照→strcpy()

206 文字列に一連の文字のいずれかの文字があるか調べる strpbrk()

ヘッダ	string.h
書式	char *strpbrk(const char **str1*, const char **str2*);
引数	*str1* ……*str2*の一連の文字のいずれかが含まれているかどうかを調べる文字列。 *str2* ……含まれるかどうか検査する文字を保存したNULLで終わるキャラクタ配列。
戻り値	文字列*str2*のいずれかの文字が*str1*に現れた位置。

● 解説

*str2*で指定した一連の文字のいずれかの文字が*str1*に含まれているかどうかを調べます。

● 例

次の例は、文字列 “Hello, good dog.” の中にある最初の「d」または「o」か「g」を探します。最初に見つかるのは、「Hello」の「o」です。

```
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *str1 = "Hello, good dog.";
   char *str2 = "dog";
   char *ptr;

   ptr = strpbrk(str1, str2);

   if (ptr)
      printf("最初の文字 %c を検出。¥n", *ptr);
   else
      printf("指定文字列の文字が見つかりません。¥n");

   return 0;
}
```

207 文字列中の文字の位置を後ろから調べる strrchr()

ヘッダ	string.h
書式	char *strrchr(const char **s*, int *c*);
引数	*s* ………文字を探す文字列。 *c* ………探す文字。
戻り値	成功したときは一致した最後の文字のポインタ、文字が見つからないときはNULL。

● 解説

文字列*s*の中に文字*c*があるかどうか、文字列の後ろから調べます。一致する最後の文字のポインタを返すこと以外、`strchr()`と同じです。

● 参照→strchr()

208 文字列から一連の文字を探す strspn()

ヘッダ	string.h
書式	size_t strspn(const char **str1*, const char **str2*);
引数	*str1* ……*str2*の一連の文字のいずれかが含まれているかどうかを調べる文字列。 *str2* ……含まれるかどうか検査する文字を保存したNULLで終わるキャラクタ配列。
戻り値	文字列*str2*のいずれかの文字だけで構成される文字列*str1*の最初の部分の長さ。

● 解説

*str2*で指定した一連の文字のいずれかが*str1*に含まれているかどうかを調べます。

209 部分文字列の位置を示す strstr()

ヘッダ	string.h
書式	char *strstr(const char **str1*, const char **str2*);
引数	*str1* ……文字列*str2*が含まれているかどうかを調べる文字列。 *str2* ……含まれるかどうか検査する文字列。
戻り値	一致する部分文字列の先頭を指すポインタ、部分文字列が見つからないときはNULL。

● 解説

文字列*str1*の中で文字列*str2*が最初に現れる位置を見つけます。

この関数のワイド文字バージョンはwchar_t *wcsstr(const wchar_t *str1, const wchar_t *str2)、マルチバイトバージョンはunsigned char *_mbsstr(const unsigned char *str1, const unsigned char *str2)です。

210 ASCII文字列をdouble型の数値に変換する strtod()

ヘッダ	stdlib.h
書式	double strtod(const char **nptr*, char ***endptr*);
引数	*nptr* ……数値に変換する文字列のポインタ。 *endptr* …数に変換できなかった最初の文字のポインタを保存するための変数のポインタ。
戻り値	文字列を変換した結果のdoubleの値。オーバーフローになった場合には正または負のHUGE_VAL。

● 解説

文字列*nptr*をdouble型の実数値に変換します。変換は現在のロケールのカテゴリLC_NUMERICの設定に基づいて行われます。*endptr*にはNULLを指定することもできます。

文字列がdouble値として変換されるためには、次のような形式になっていなければならず、double値として解釈できない文字があると、そこで変換を中止します。

[ws] [sn] [ddd] [.] [ddd] [fmt[sn]ddd]

各部分の詳細は以下のとおりです。

要素	意味
[ws]	ホワイトスペース（なくてもよい）。
[sn]	符号（+または-，なくてもよい）。
[ddd]	数字列（なくてもよい）。
[fmt]	オプションのeまたはE（なくてもよい）。
[.]	小数点（なくてもよい）。

+INFと-INFはプラス無限大とマイナス無限大として認識します。+NANと-NANを非数（Not-a-Number）として認識します。

*endptr*がNULLでなければ、スキャンを中止した文字を指すポインタを*endptrにセットします。これはエラーを検出するときに使うことができます。

● 例

次の例は入力された文字列を数値に変換します。入力された文字列が数字以外の文字で始まる場合はそのことを検出して表示します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ], *endptr;
   double v;

   /* プログラムのメインロジック */
   printf("文字列を入力してください>");
   scanf("%s", buff);

   /* 実数に変換する */
   v = strtod(buff, &endptr);
   if (endptr == &buff[0])
      printf("文字列% s は数値ではありません.¥n", buff);
   else
      printf("値は=%f¥n", v);

   return 0;
}
```

● **参照**→atof()、setlocale()

211 文字列から次のトークンを取り出す strtok()

ヘッダ	string.h
書式	char *strtok(char **str*, const char **delim*);
引数	*str*………トークンの入っている文字列。 *delim* …デリミタ文字の配列のポインタ。
戻り値	*str*にある次のトークンのポインタ。それ以上トークンがないときはNULL。

● 解説

*str*の中で、*delim*の各文字をデリミタ（区切り文字）として次のトークンを見つけます。2回目以降の呼び出しでは*str*にNULLを指定すると、すでにトークンを取り出した部分を除いた残りの部分から次のトークンを取り出します。

●例

次の例は、標準入力から入力された文字列を、区切り文字（スペース　,．¥t、¥n）で切り分けて出力します。

```
#include <string.h>
#include <stdio.h>

#define BUFFSIZE 1024

int main(int argc, char *argv[])
{
   char buff[BUFFSIZE];
   char delim[] = " ,.¥t¥n";
   char *tok;

   printf("文字列>");
   fgets(buff, BUFFSIZE, stdin);

   /* 最初のトークンを取得する */
   tok = strtok(buff, delim);
   while(tok != NULL)
   {
      /* 取り出したトークンを出力する */
      printf("%s¥n", tok);
      /* 次のトークンを取得する */
      tok = strtok(NULL, delim);
   }
   return 0;
}
```

212 文字列をロング整数に変換する strtol()

ヘッダ	stdlib.h
書式	long int strtol(const char **nptr*, char ***endptr*, int *base*);
引数	*nptr* ……変換する文字列。 *endptr*…変換のスキャンを終了した位置を示す文字へのポインタ。 *base* ……変換する際の数の基数。
戻り値	変換された文字列の値、エラーが発生したときは0。

解説

文字列*nptr*をlong値に変換します。この関数は認識できない最初の文字があるとそこで文字列の読み出しを中止し、そのポインタを*endptr*に保存します。

変換する文字列は次の書式に一致しなければなりません。

[ws] [sn] [0] [x] [ddd]

各部分の詳細は以下の表のとおりです。

要素	意味
[ws]	ホワイトスペース（なくてもよい）。
[sn]	符号（+または-，なくてもよい）。
[0]	0（なくてもよい）。
[x]	xまたはX（なくてもよい）。
[ddd]	数字列（なくてもよい）。

*base*が2と36の範囲内であれば、基数が*base*の値として変換されます。*base*が0の場合、*nptr*の最初の2〜3文字によって変換される値の基数が決まります。

▼表　baseが0の場合の文字列の解釈

第1字	第2字	文字列の解釈
0	1〜7	8進数
0	xまたはX	16進数
1〜9		10進数

*base*が1、負あるいは36を超える値であると無効な値とみなされます。

例

次の例は、文字列を数値に変換するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ], *endptr;
   long v;

   printf("文字列を入力してください>");
   scanf("%s", buff);

   v = (int)strtol(buff, &endptr, 10);

   if (endptr == &buff[0])
      printf("文字列% s は数値ではありません.¥n", buff);
   else
      printf("値は=%ld¥n", v);

   return 0;
}
```


213 文字列を符号なしロング整数に変換する strtoul()

ヘッダ	stdlib.h
書式	unsigned long int strtoul(const char **nptr*, char ***endptr*, int *base*)
引数	*nptr* ……変換する文字列。 *endptr* …変換のスキャンを終了した位置を示す文字へのポインタ。 *base* ……変換する際の数の基数。
戻り値	変換された値、エラーが発生したときは0。

● 解説

文字列を符号なしロング整数（unsigned long）に変換します。

● 例

次の例は、文字列を数値に変換するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
   char buff[BUFSIZ], *endptr;
   long v;
   printf("文字列を入力してください>");
   scanf("%s", buff);
   v = (int)strtoul(buff, &endptr, 10);
   if (endptr == &buff[0])
      printf("文字列% s は数値ではありません.¥n", buff);
   else
      printf("値は=%ld¥n", v);

   return 0;
}
```

● **参照**→strtol()

214 ロケール固有情報に基づいて文字列を変換する strxfrm()

ヘッダ	string.h
書式	size_t strxfrm(char **dest*, const char **src*, size_t *n*);
引数	*dest* ……変換先の文字列。 *src*………変換元の文字列。 *n* ………destの最大文字数（終端のNULL文字を除く）。
戻り値	変換された文字列の長さ。戻り値がn以上の場合は*dest*の内容は不定。 エラーが発生したときは-1。

● **解説**

文字列*src*を現在のロケールのカテゴリLC_COLLATEの設定に従って変換します。

215 隣接するバイトを交換する swab()

ヘッダ	string.h
書式	void swab(const void **src*, void **dest*, int *n*);
引数	*src*………コピー後スワップされるデータ。 *dest* ……スワップされたデータの格納領域。 *n* ………コピー後スワップするバイト数。

● 解説

配列*src*から*n*バイトを*dest*にコピーして、隣接した偶数/奇数バイトを交換します。バイトオーダが異なるマシン間でのデータ交換に使うと便利です。

● 例

次の例は文字列を2文字ずつ交換するプログラムの例です。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char src[] = "abcdefgh";
char target[15];

int main(void)
{
   printf("swab前=%s¥n", src);

#ifdef WIN32
   _swab(src, target, strlen(src));
#else
   swab(src, target, strlen(src));
#endif

   printf("swab後=%s¥n", target);

   return 0;
}
```

実行結果は以下のとおりです。

```
swab前=abcdefgh
swab後=badcfehg
```

216 隣接するバイトを交換する _swab()

ヘッダ	string.h
書式	void _swab(const void **src*, void **dest*, int *n*);
引数	*src*………コピー後スワップされるデータ。 *dest* ……スワップされたデータの格納領域。 *n* ………コピー後スワップするバイト数。

●解説

配列*src*から*n*バイトを*dest*にコピーして、隣接した偶数/奇数バイトを交換します。

●準拠… ISO C++

●参照… swab()

217 コマンドを実行する system()

ヘッダ	stdlib.h
書式	int system (const char * *str*);
引数	*str*………実行するコマンド文字列。
戻り値	エラーが発生したときは-1。それ以外の戻り値は環境に依存。

●解説

コマンド*str*を実行します。この関数の動作や実行した結果の詳細、戻り値などは、OSやコマンドインタプリタによって異なることがあります。

●例

次の例は、現在のディレクトリの内容を表示するプログラムの例です。

```
#include <stdlib.h>

int main( void )
{
#ifdef WIN32
   system( "dir" );
#else
   system( "ls -la" );
#endif
}
```

218 タンジェントを計算する tan()

ヘッダ	math.h
書式	double tan(double *x*);
引数	*x* ………タンジェントを計算したい値（ラジアン単位）。
戻り値	タンジェントの値。

● **解説**

*x*のタンジェント（正接）の値を計算します。

● **参照**→sin()

219 ハイパーボリックタンジェントを計算する tanh()

ヘッダ	math.h
書式	double tanh(double *x*);
引数	*x* ………ハイパーボリックタンジェントを計算したい値（ラジアン単位）。
戻り値	ハイパーボリックタンジェントの値。

● **解説**

*x*のハイパーボリックタンジェント（双曲線正接）を計算します。この値はsinh(x) / cosh(x)です。

● **参照**→tan()、cosh()、sinh()

220 テンポラリファイルを作成する tmpfile()

ヘッダ	stdio.h
書式	FILE *tmpfile (void);
戻り値	成功するとファイルポインタ、ユニークなファイルが作れなかったかオープンできなかったときはNULL。

● **解説**

ユニークなファイルをバイナリリードライトモード(w+b)で作成します。

このファイルは、クローズしたときまたはプログラムの終了時に自動的に削除されます。

● **エラーコード**

EACCES、EEXIST、EMFILE、ENFILE、EROFS

221 テンポラリファイル名を生成する tmpnam()

ヘッダ	stdio.h
書式	char *tmpnam(char *str);
引数	*str*………テンポラリファイル名のポインタ。
戻り値	ユニークな名前を生成したときは生成した名前のポインタ。 名前を作成できないときやその名前を持つファイルがすでにある場合はNULL。

●解説

既存のファイルに上書きせずにテンポラリファイルを開くための一時的な名前を生成します。テンポラリファイルを作るときに同じ名前のファイルがあるかどうか調べたいときに便利です。

●互換性

同等の機能を持つ関数として以下のものがあります。

```
char *_tempnam(char *dir char *prefix)
wchar_t *_wtempnam(wchar_t *dir, wchar_t *prefix)
wchar_t *_wtmpnam(wchar_t *str)
```

222 大文字を小文字にする tolower()

ヘッダ	ctype.h
書式	int tolower (int *c*);
引数	*c* ………小文字に変換する文字。
戻り値	変換できれば変換後の文字、できなければ変換前の文字。

●解説

*c*が小文字にできる文字であるならば小文字に変換します。変換の結果は現在のロケールに依存します。

●例

次の例は、文字列の中の大文字をすべて小文字にします。

```
#include <stdio.h>
#include <ctype.h>

int main( void )
{
   int i;
   char s[] = "AbCdEfGh";

   puts(s);

   for (i=0; i<strlen(s); i++)
      if (isupper((int)s[i]))
         s[i] = tolower(s[i]);

   puts(s);
}
```

実行結果は以下のとおりです。

```
AbCdEfGh
abcdefgh
```

● **参照**→toupper()

223 大文字を小文字にする _tolower()

ヘッダ	ctype.h
書式	int _tolower (int *c*);
引数	*c* ………小文字に変換する文字。
戻り値	変換できれば変換後の文字、できなければ変換前の文字。

● **解説**

*c*が小文字にできる文字であるならば小文字に変換します。

● **参照**→tolower()

224	小文字を大文字にする toupper()
ヘッダ	ctype.h
書式	int toupper (int *c*);
引数	*c* ………大文字に変換する文字。
戻り値	変換できれば変換後の文字、できなければ変換前の文字。

解説

*c*が大文字にできる文字であるならば大文字に変換します。変換の結果は現在のロケールに依存します。

参照→tolower()

225	小文字を大文字にする _toupper()
ヘッダ	ctype.h
書式	int _toupper (int *c*);
引数	*c* ………大文字に変換する文字。
戻り値	変換できれば変換後の文字、できなければ変換前の文字。

解説

*c*が大文字にできる文字であるならば大文字に変換します。

226	大文字を小文字にする towlower()
ヘッダ	ctype.h
書式	int towlower (wint_t *c*);
引数	*c* ………小文字に変換する文字。
戻り値	変換できれば変換後の文字、できなければ変換前の文字。

解説

ワイド文字*c*が小文字にできる文字であるならば小文字に変換します。

参照→tolower()

227 ファイルを指定した長さに切り詰める truncate()

ヘッダ	unistd.h
書式	int truncate(const char **path*, size_t *len*);
引数	*path* ……長さを変更するファイルのパス。 *len* ……切り詰め後のファイルの長さ。
戻り値	成功したときは0、失敗したときは-1。

● 解説

ファイル*path*を*len*バイトの長さになるように延長もしくは切り詰めます。

228 時刻の変換情報を初期化する tzset()

ヘッダ	time.h
書式	void tzset (void);

● 解説

環境変数TZの現在の設定を使って、時刻の変換情報に関連する変数の値を初期化します。

● 例

asctime()、strftime()の例参照

229 時刻の変換情報を初期化する _tzset()

ヘッダ	time.h
書式	void _tzset (void);

● 解説

環境変数TZの現在の設定を使って、時刻の変換情報に関連する変数の値を初期化します。

● 例

asctime()、strftime()の例参照

● **参照**→tzset()

230 ファイル作成マスクをセットする umask()

ヘッダ	sys/stat.h
書式	int umask(int *mask*);
引数	*mask* …ファイルのアクセス権を指定するためのマスク値。
戻り値	それまでのマスク値。

● 解説

ファイルを新しく作成するときの、デフォルトのファイルアクセス権を設定します。ファイルのアクセス権はファイルシステムによって異なるので、この関数の動作の詳細もファイルシステムによって異なります。

231 ファイル作成マスクをセットする _umask()

ヘッダ	io.h、sys/stat.h、sys/types.h
書式	int _umask(int pmode);

● 参照→umask()

232 文字をファイルに戻す ungetc()

ヘッダ	stdio.h
書式	int ungetc(int *c*, FILE **fp*);
引数	*c* ………ファイルに戻す文字。 *fp* ………文字を戻すファイル。
戻り値	成功したときは*c*、エラーが発生したときはEOF。

● 解説

その後の読み込み操作で読み込めるように、文字*c*を`unsigned char`にキャストして*fp*に書き戻します。

233 名前やファイルを削除する unlink()

ヘッダ	unistd.h
書式	int unlink(const char **path*);
引数	*path* ……削除する名前。
戻り値	成功したときは0、エラーが発生したときは-1。

● 解説

ファイルシステム上の名前（ファイル名やシンボリックリンク）を削除します。

● エラーコード

EACCES、EFAULT、EIO、EISDIR、ELOOP、ENAMETOOLONG、ENOENT、ENOMEM、ENOTDIR、EPERM、EROFS

● 注意

NFSではまだ使っているファイルが突然消滅することがあります。

● 互換性

同等の機能の関数として、`int _unlink(const char *path)`と`int _wunlink(const wchar_t *path)`があります。

234 環境変数を削除する unsetenv()

ヘッダ	stdlib.h
書式	void unsetenv(const char *name);
引数	*name* …削除する環境変数名。

● 解説

環境変数*name*を環境から削除します。

● 参照→setenv()

235 ファイルのタイムスタンプを設定する utime()

ヘッダ	sys/types.h、utime.h
書式	int utime(const char *fname, struct utimbuf *buf);
引数	*fname* …タイムスタンプを変更するファイル名。 *buf* ……utimbuf構造体の変数のポインタ。
戻り値	成功したときは0、失敗したときは-1。

● 解説

ファイル*fname*のタイムスタンプ（inodeのアクセス時間と更新時間）をutimbuf構造体変数*buf*のメンバactimeとmodtimeにそれぞれ変更します。*buf*がNULLの場合はファイルのアクセス時間と更新（変更）時間は現在の時間にセットされます。utimbuf構造体は次のように定義されています。

```
struct utimbuf {
    time_t actime; /* アクセス時刻 */
    time_t modtime; /* 更新時刻 */
};
```

● **エラーコード**… EACCES、ENOENT

● **互換性**

この関数はファイルシステムに依存します。同等機能の関数として、int _utime(unsigned char *fname, struct _utimbuf *times)と、int _wutime(wchar_t *fname, struct _utimbuf *times)があります。

236 ファイルのタイムスタンプを変更する utimes()

ヘッダ	sys/types.h、sys/time.h
書式	int utimes(char **fname*, struct timeval **ptv*);
引数	*fname* …タイムスタンプを変更するファイル名。 *ptv* ……timeval構造体の変数のポインタ。
戻り値	成功したときは0、失敗したときは-1。

● **解説**

ファイル*fname*のタイムスタンプ（inodeのアクセス時間と更新時間）をtimeval構造体変数*ptv*の値に従って変更します。

timeval構造体は次のように定義されています。

```
struct timeval {
    long tv_sec; /* 秒 */
    long tv_usec; /* ミリ秒 */
};
```

● **互換性**

この関数はtvp[0].tv_secをactimeとして、tvp[1].tv_secをmodtimeとして使ってutime()を呼び出すように実装されていることがあります。

● **参照**→utime()

237 個数が変わる引数リストを提供する va_arg()

ヘッダ	stdarg.h
書式	type va_arg(va_list *arg_ptr*, *type*);
引数	*arg_ptr*……引数リストへのポインタ。 *type* ………取得する引数の型。
戻り値	引数リストのカレント引数。

●解説

va_arg()は、va_start()やva_end()などと共に使って、Cの関数で可変個の引数リストを実現します。最初にva_arg()を使ったときには、引数リストの最初の引数が返されます。その後，va_arg()を使うたびに引数リストの次の引数が返されます。

va_start()、va_arg()、va_end()は必ずこの順序で使います。またこれらは、通常は関数ではなくマクロとして実装されています。

238 個数が変わる引数リストへのアクセスを終了する va_end()

ヘッダ	stdarg.h
書式	void va_end(va_list *arg_ptr*);
引数	*arg_ptr* ………引数リストのポインタ。

●解説

va_arg()がすべての引数を読み終わった後で呼び出し、呼び出した関数が通常のリターン動作を行えるようにします。

●参照→va_arg()

239 個数が変わる引数リストへのアクセスを開始する va_start()

ヘッダ	stdarg.h
書式	① void va_start(va_list *arg_ptr*, *prev*); ② void va_start(va_list *arg_ptr*);
引数	*arg_ptr*……引数リストのポインタ。 *prev* ………最初のオプション引数の直前の引数。

● 解説

*arg_ptr*が、その関数に渡される第1の可変個の引数の先頭を指すようにセットします。va_arg()を最初に呼び出す前にva_start()を呼び出さなければなりません。

書式①はANSIバージョンの書式、書式②は以前のUNIXバージョンのva_start()の書式です。

● 参照→va_arg()

240 引数リストの値をファイルに出力する vfprintf()

ヘッダ	stdio.h、stdarg.h
書式	int vfprintf(FILE **fp*, const char **fmt*, va_list *arg_ptr*);
引数	*fp* …………出力するファイル。 *fmt* ………書式指定。printf()参照。 *arg_ptr*……引数リストのポインタ。
戻り値	出力した文字の数。

● 解説

ファイル*fp*に引数リスト*arg_ptr*の値を書式*fmt*に従って出力します。

● 参照→printf()

241 引数リストの値を標準出力に出力する vprintf()

ヘッダ	stdio.h、stdarg.h
書式	int vprintf(const char **fmt*, va_list *arg_ptr*);
引数	*fmt* …………書式指定。printf()参照。 *arg_ptr* ……引数リストのポインタ。
戻り値	出力した文字の数。

● 解説

標準出力（stdout）に引数リスト*arg_ptr*の値を書式*fmt*に従って出力します。

● 参照→printf()

242 引数リストの値を文字列バッファに出力する vsprintf()

ヘッダ	stdio.h、stdarg.h
書式	int vsprintf(char **str*, const char **fmt*, va_list *arg_ptr*);
引数	*str* …………出力する文字列バッファのポインタ。 *fmt* …………書式指定。printf()参照。 *arg_ptr* ……引数リストのポインタ。
戻り値	出力した文字の数。

● **解説**

文字列バッファ*str*に引数リスト*arg_ptr*の値を書式*fmt*に従って出力します。

● **参照**→printf()

243 ファイルのアクセス権をチェックする _waccess()

ヘッダ	wchar.hまたはio.h
書式	int _waccess(const wchar_t **path*, int *mode*);
引数	*path* ……アクセス権を調べるファイル名。 *mode* …ファイルのモード。解説の表に示す値のいずれか。
戻り値	要求がすべて満たされる場合は0。そうでなければ-1。

● **解説**

指定したファイルに対して、読み出しや書き込みが許されているか調べます。

● **準拠**… ISO C++

● **参照**→access()

244 カレントディレクトリを変更する _wchdir()

ヘッダ	direct.h または wchar.h
書式	int _wchdir(const wchar_t **path*);
引数	*path* ……変更後の新しいパスを表す文字列定数。
戻り値	成功したときは0、失敗すると-1。

● **解説**

カレントディレクトリを、指定したディレクトリ*path*に変更します。

● **準拠**… ISO C++

● **参照**→chdir()

245 ファイルのアクセスモードを変更する _wchmod()

ヘッダ	io.h、sys/stat.h、sys/types.h
書式	int _wchmod(const wchar_t **filename*, int *amode*);
引数	*filename* ……アクセスモードを変更するファイル名。 *amode* ………アクセス許可を以下のフラグの組み合わせで指定します。
戻り値	成功したときは0。失敗すると-1。

● **解説**

ファイルのアクセスモードを変更します。アクセスモードには、_S_IWRITE（読み書き許可）または_S_IREAD（読み出し許可）を指定します。

● **準拠**… ISO C++

● **参照**→chmod()

246 ディレクトリを閉じる（ワイド文字バージョン） wclosedir()

ヘッダ	sys/types.h、dirent.h
書式	int wclosedir(DIR **dir*);
引数	*dir*………閉じるディレクトリのディレクトリストリームデスクリプタ（は通常wopendir()が返した値）。
戻り値	成功したときは0、失敗したときは-1。

● **解説**… *dir*に連結しているディレクトリストリームを閉じます。

● **参照**→closedir()

247 ファイルかデバイスを作成する _wcreat()

ヘッダ	io.h または wchar.h
書式	int _wcreat(const wchar_t **path*, int *mode*);
引数	*path*……オープンまたは作成するファイルまたはデバイスの名前。 *mode* …ファイルまたはデバイスをオープンまたは作成するときのモード（open()参照）。
戻り値	成功したときは新しいファイルデスクリプタ、エラーが発生したときは-1。

● 解説

ファイルまたはデバイスを作成します。

● 準拠… ISO C++

● 参照→creat()

248 ワイド文字の文字列を比較する wcscmp()

ヘッダ	string.h
書式	int wcscmp(const wchar_t **s1*, const wchar_t **s2*);
引数	*s1*、*s2* …比較する文字列。
戻り値	比較した結果を示す整数（解説参照）。

● 解説

文字列*s1*と*s2*を比較します。

戻り値の意味は次の表に示すとおりです。

▼表 wcscmp()の戻り値の意味

返される値	文字列s1とs2の関係
0未満の値	s1がs2より小さい。
0	s1とs2が等しい。
0を超える値	s1がs2より大きい。

● 参照→strcmp()

249	日付と時間を書式化する wcsftime()
ヘッダ	time.h
書式	size_t wcsftime(wchar_t **s*, size_t *max*, const wchar_t **fmt*, const struct tm **t*);
引数	*s* ………書式化した文字列を保存するバッファのポインタ。 *max* ……文字列の最大の長さ。 *fmt* ……書式制御文字列。 *t* ………tmデータ構造体。
戻り値	成功したときはバッファに保存した文字列の文字数、エラーが発生したときは**0**。

● 解説

tmデータ構造体のポインタptmが指す時刻を書式*fmt*に従って書式化し、文字列にしてsに保存します。

● 参照→strftime()

250	ワイド文字の文字列の長さを取得する wcslen()
ヘッダ	string.h
書式	size_t wcslen(const wchar_t **str*);
引数	*str* ………長さを調べる文字列。
戻り値	文字列の長さ（文字単位）。

● 解説

文字列の長さを文字単位で取得します。

● 例

次の例は、ワイド文字の長さを調べて表示するプログラムの例です。

```
#include <stdio.h>
#include <locale.h>
#include <string.h>

int main( void )
{
   int l;
   wchar_t ws[] = L"いろはにほへと";

   setlocale(LC_ALL, "japanese");

   l = wcslen(ws);

   wprintf(L"(%d):%s¥n", l, ws);
}
```

実行結果は以下のとおりです。

```
(7):いろはにほへと
```


● **参照**→strlen()

251 文字列に一連の文字のいずれかの文字があるか調べる _wcspbrk()

ヘッダ	string.h
書式	wchar_t * _wcspbrk(const wchar_t **str1*, const wchar_t **str2*);
引数	*str1*…*str2*の一連の文字のいずれかが含まれているかどうかを調べる文字列。 *str2*…含まれるかどうか検査する文字を保存したNULLで終わるキャラクタ配列。
戻り値	文字列*str2*のいずれかの文字が*str1*に現れた位置。

● **解説**

*str2*で指定した一連の文字のいずれかの文字が*str1*に含まれているかどうかを調べます。

● **参照**→strpbrk()

252 ワイド文字列をdouble型の数値に変換する wcstod()

ヘッダ	stdlib.h
書式	double wcstod(const wchar_t **nptr*, wchar_t ***endptr*);
引数	*nptr* ……数値に変換する文字列のポインタ。 *endptr* …数に変換できなかった最初の文字のポインタを保存するための変数のポインタ。
戻り値	文字列を変換した結果のdoubleの値。 オーバーフローになった場合には正または負のHUGE_VAL。

● 解説

文字列*nptr*を`double`型の実数値に変換します。

● 参照→strtod()

253 ワイド文字列をロング整数に変換する wcstol()

ヘッダ	stdlib.h
書式	long int wcstol(const char **nptr*, char ***endptr*, int *base*);
引数	*nptr* ……変換する文字列。 *endptr* …変換のスキャンを終了した位置を示す文字へのポインタ。 *base* ……変換する際の数の基数。
戻り値	変換された文字列の値、エラーが発生したときは0。

● 解説

文字列*nptr*を`long`値に変換します。

● 参照→strtol()

254 ワイド文字列をマルチバイト文字列に変換する wcstombs()

ヘッダ	stdlib.h
書式	size_t wcstombs(char **mbstr*, const *wcstr*_t *wcstr, size_t *n*);
引数	*mbstr* …変換したマルチバイト文字列を入れるバッファ。 *wcstr* …変換元のワイド文字列。 *n* ………出力先のマルチバイト文字列に保存できる最大バイト数。
戻り値	保存されたバイト数。 ワイド文字列に無効なワイド文字が含まれるときは-1。

●解説

ワイド文字列*wcstr*をマルチバイト文字列に変換して*mbstr*に最大*n*バイトを保存します。

●参照→mbstowcs()

255 ワイド文字列を符号なしロング整数に変換する wcstoul()

ヘッダ	stdlib.h
書式	unsigned long int wcstoul(const char **nptr*, char ***endptr*, int *base*)
引数	*nptr* ……変換する文字列。 *endptr* …変換のスキャンを終了した位置を示す文字へのポインタ。 *base* ……変換する際の数の基数。
戻り値	変換された値、エラーが発生したときは0。

●解説

文字列を符号なしロング整数（unsigned long）に変換します。

●参照→strtoul()

256 時刻のバイナリデータをワイド文字列に変換する wctime()

ヘッダ	time.h
書式	wchar_t *_wctime(const time_t **timep*);
引数	*timep* …文字列に変換したい時刻の情報が入っている構造体のポインタ。
戻り値	時刻を表す文字列のポインタ。

● 解説

time_t型のカレンダー時刻を文字列に変換します。

● 参照→ctime()

257 ワイド文字をマルチバイト文字列に変換する wctomb()

ヘッダ	stdlib.h
書式	int wctomb(char **mbstr*, wchar_t *wcstr*);
引数	*mbstr* …変換したマルチバイト文字列を入れるバッファ。 *wcstr* …変換元のワイド文字列。
戻り値	保存されたバイト数。 ワイド文字列に無効なワイド文字が含まれるときは-1。

● 解説

ワイド文字列*wcstr*をマルチバイト文字列に変換して*mbstr*に保存します。

● 参照→mbtowc()

258 プログラムを実行する _wexecl()

ヘッダ	process.h
書式	intptr_t intptr_t _wexecl(const wchar_t **path*, const wchar_t **arg*, ...);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *arg* ……プログラムの引数（任意の数だけ指定）。
戻り値	この関数がリターンしたときは戻り値は-1（エラーが発生している）

● 解説

指定された実行可能なプログラムを実行します。

● 参照→_execl()、execl()

259 プログラムを実行する _wexecle()

ヘッダ	process.h
書式	int _wexecle(const wchar_t **path*, const wchar_t **arg* , ..., wchar_t * const *envp*[]);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *arg* ……プログラムの引数を任意の数だけ指定します。 引数リストの最後はNULLでなければなりません。 *envp*……環境設定の配列のポインタ。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● 解説

指定されたプログラムを、指定された環境で実行します。

● 参照→execle()

260 プログラムを実行する _wexeclp()

ヘッダ	process.h
書式	int _wexeclp(const wchar_t **file*, const wchar_t **arg*, ...);
引数	*file* ……実行するファイル名。 *arg* ……実行するファイルに渡す引数。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● **解説**

指定されたファイルを実行します。

● **参照**→execlp()

261 プログラムを実行する _wexecv()

ヘッダ	process.h
書式	int _wexecv(const wchar_t **path*, const wchar_t *argv*[]);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *argv* ……プログラムの引数の配列のポインタ（配列の最後の要素はNULL）。
戻り値	この関数がリターンしたときは-1（エラーが発生している）。

● **解説**

指定されたプログラムを、配列で指定された引数を伴って実行します。

● **参照**→execv()

262 プログラムを実行する _wexecve()

ヘッダ	process.h
書式	int _wexecve(const wchar_t **cmdname*, const wchar_t **argv*, const wchar_t **envp*);
引数	*cmdname* …実行するプログラムを示す文字列のポインタ。 *argv* ………プログラムの引数の配列のポインタ(配列の最後の要素はNULL)。 *envp* ………環境設定の配列のポインタ（配列の最後の要素はNULL)。
戻り値	この関数がリターンしたときは-1（エラーが発生している)。

解説

指定されたプログラムを、指定された環境で実行します。

参照→execve()

263 プログラムを実行する _wexecvp()

ヘッダ	process.h
書式	int _wexecvp(const wchar_t **path*, const wchar_t *argv*[]);
引数	*path* ……実行するプログラムを示す文字列のポインタ。 *argv* ……プログラムの引数の配列のポインタ（配列の最後の要素はNULL)。
戻り値	この関数がリターンしたときは-1（エラーが発生している)。

解説

指定されたプログラムを指定された環境で実行します。

参照→execvp()

264 カレントディレクトリ名を取得する _wgetcwd()

ヘッダ	wchar.h
書式	char *getcwd(wchar_t **buf*, size_t *size*);
引数	*buf* ……ディレクトリパスを保存するバッファのアドレス。NULLも指定可。 *size* ……バッファの長さ。
戻り値	成功したときは取得した絶対パスのポインタ。 エラーが発生したときはNULL。

● 解説

カレントディレクトリの絶対パス名をバッファ*buf*に取得します。

● 準拠… ISO C++

● 参照→getcwd()

265 ディレクトリを作成する _wmkdir()

ヘッダ	direct.h、wchar.h
書式	int wmkdir(const wchar_t **path*);
引数	*path* ……作成するディレクトリのパス。
戻り値	新しいディレクトリが作成されたときは0、エラーのときは-1。

● 解説

ディレクトリを作成します。

● 参照→mkdir()

266 ファイルやデバイスをオープンする _wopen()

ヘッダ	io.h、wchar.h
書式	int _wopen(const wchar_t **path*, int *flags* [, int *mode*]);
引数	*path*……オープンするファイルやデバイスのパス。 *flags*……オープンするファイルに許可する操作を指定するフラグ。 *mode* …オープンするファイルのモード。省略可能です。
戻り値	新しいファイルデスクリプタ、エラーが発生したときは-1。

● 解説

ファイルのオープンを試み、ファイルデスクリプタを返します。

● **参照**→open()

267 パイプを作成してコマンドを実行する _wpopen()

ヘッダ	stdio.h
書式	FILE *_popen(const wchar_t **command*, const wchar_t **type*);
引数	*command* …実行するコマンド。 *type*…………返されるストリームの種類（読み出しは'r'、書き込みは'w'）。
戻り値	正常に終了できたときはファイルポインタ、そうでないときはNULL。

● 解説

パイプを作成してコマンドを実行します。

● **参照**→popen()

268 ワイド文字を出力する wprintf()

書式	int wprintf(const wchar_t **fmt* [,argument, ...]);
引数	*fmt* ……出力するときの書式。printf()の解説参照。 ... ………出力する値のリスト。
戻り値	成功したときは出力したバイト数、エラーの場合はEOF。

● 解説

printf()のワイド文字バージョンです。

● **参照**→printf()

269 ファイルを削除する _wremove()

ヘッダ	stdio.h、io.h
書式	int _wremove(const wchar_t **path*);
引数	*path* ……削除するファイルのパス名。
戻り値	成功したときは0、失敗したときは-1。

● 解説

*path*で指定されたファイルを削除します。ファイルを削除する前にそのファイルのハンドルをすべて閉じる必要があります。

● 参照→remove()

270 データをファイルに書き込む write()

ヘッダ	unistd.h
書式	int write(int *fd*, const void **buf*, unsigned int *n*);
引数	*fd* ………データを書き込むファイルのデスクリプタ。 *buf* ……書き込むデータが入っているバッファのポインタ。 *n* ………書き込むバイト数。
戻り値	成功したときは書き込まれたバイト数、エラーが発生したときは-1。

● 解説

バッファ*buf*のデータ*n*バイトを*fd*で識別されるファイルに書き込みます。

● エラーコード… EAGAIN、EBADF、EFAULT、EINTR、EINVAL、EIO、EPIPE

271 データをファイルに書き込む _write()

ヘッダ	unistd.h
書式	int _write(int *fd*, const void **buf*, unsigned int *n*);
引数	*fd* ………データを書き込むファイルのデスクリプタ。 *buf* ……書き込むデータが入っているバッファのポインタ。 *n* ………書き込むバイト数。
戻り値	成功したときはには書き込まれたバイト数、エラーが発生したときは-1。

解説… バッファ*buf*のデータ*n*バイトを*fd*で識別されるファイルに書き込みます。

参照→write()

準拠… ISO C++

272 ファイルについての情報を取得する _wstat()

ヘッダ	sys/types.hとsys/stat.h
書式	int _wstat(const wchar_t **path*, struct _stat **buf*);
引数	*path* ……情報を取得するファイルのパス。 *buf* ……取得した情報を保存する_stat構造体変数のポインタ。
戻り値	成功したときは0、エラーが発生したときは-1。

解説… ファイルまたはディレクトリに関する情報を取得します。

参照→stat()

273 コマンドを実行する _wsystem()

ヘッダ	stdlib.h
書式	int _wsystem (const wchar_t * *str*);
引数	*str* ………実行するコマンド文字列。
戻り値	エラーが発生したときは-1。それ以外の戻り値は環境に依存。

解説… コマンド*str*を実行します。

参照→system()

エラーコード

ライブラリ関数が返す可能性がある代表的なエラーコードの一覧を示します。
実際に返されるコードの種類や厳密な定義は、環境によって異なることがあります。

▼表 エラーコード

エラーコード	定義
EACCES	パスにアクセスできない。指定したパスに対するアクセスや検索の許可がないか、適切なデスクリプタが指定されていない。
EADDRINUSE	アドレスがすでに使われている。
EALREADY	ソケットが非停止に設定されていて、以前の接続が完了していない。
EBADF	ファイルデスクリプタが無効。あるいは書き込み操作で書き込みのためにオープンされていない。
ECONNREFUSED	サーバーによって接続が拒否された。
EDOM	引数の値が不正（範囲外）。
EEXIST	指定されたパス名のファイルがすでに存在している。
EFAULT	アクセス可能なアドレス空間の外を指している。
EINPROGRESS	ソケットが非停止に設定されていて、接続がすぐに完了しない。
EINTR	操作の前にシグナルによって割り込まれた。
EINVAL	引数に不正な値が指定された。あるいは結果を保存するバッファが小さすぎる。
EIO	I/Oエラーが発生した。
EISCONN	ソケットはすでに接続されている。
EISDIR	デスクリプタまたはパスがディレクトリを参照している。
ELOOP	パス名を解決する際に遭遇するシンボリックリンクが多すぎる。
EMFILE	システムで開かれているファイルが多すぎるか、プロセスで使われているファイルデスクリプタが多すぎる。
ENAMETOOLONG	ファイル名やそのほかの名前が長すぎる。
ENETUNREACH	ネットワークが到達できない。
ENFILE	システム全体でオープン可能なファイルの限界に達している。
ENOENT	ファイルやディレクトリが存在しないか、シンボリックリンクが壊れている。
ENOMEM	メモリが足りない。
ENOSPC	ファイルまたはデバイスに十分な空きがない。
ENOTDIR	パス名のディレクトリ部分が、実際にはディレクトリでない。あるいは、ファイルデスクリプタがディレクトリを参照していない。

03-04 エラーコード

ENOTSOCK	デスクリプタはソケットではない。
EOPNOTSUPP	参照しているソケットの型がSOCK_STREAMではない。
EPERM	さまざまな許可違反。許可されていないオブジェクトにアクセスしようとした。
EPIPE	パイプかソケットに接続されていて、その反対側が閉じている。
EROFS	ファイルやソケットが読み出し専用のファイルシステムに対して書き込みアクセスを要求した。
ESRCH	プロセスIDがどのプロセスとも一致しない。
ETIMEDOUT	接続を試みている途中でタイムアウトになった。
ETXTBSY	パスが現在実行中の実行イメージを参照していて、書き込みを要求した。
EWOULDBLOCK	ソケットは非ブロック化されていて、受け付ける接続要求がない。

04：C++
リファレンス

ここでは標準C++の定数や標準C++ライブラリに含まれるプログラミング要素について説明します。C++のシンタックスはC言語によく似ているので「02 C/C++のシンタックス」で説明しています。

01 標準C++ライブラリ概要

標準C++ライブラリは、一般的な用途に効果的に利用できる汎用クラスと関数を集めたライブラリです。標準C++ライブラリの要素は以下のとおりです。

- 標準テンプレートライブラリ（STL）
- IOStream機能
- ロケール機能
- テンプレート化された文字列クラス（stringクラス）
- 複素数を表現するテンプレート化された複素数クラス（complexクラス）
- 数値配列処理用に最適化された数値の配列クラス（valarrayクラス）
- 基本データ型の値の範囲や符号などの情報を統一した方法で提供するクラス（numeric_limitsクラス）
- メモリ管理機能
- 多言語文字セットのサポート
- 例外処理機能

STL

STL（Standard Template Library、標準テンプレートライブラリ）は、さまざまなオブジェクトを保存するコンテナと、それにアクセスするときに使う反復子（iterator）、コンテナの内容を操作するためのアルゴリズムからなります。

コンテナ

STLコンテナは、オブジェクトを保存するために使います。たとえば、プログラムの中で名前のリストを使いたいとします。STLを使って次のようにすると、stringクラスのリストを簡単に作ることができます。

```
#include <string>
#include <list>

using namespace std;

list<string> namelist;
```

そして、たとえば、listのメンバ関数push_back()を使って次のようにすれば、任意の長さの名前をリストnamelistに保存できます。

```
string name; // 名前が入っている変数
namelist.push_back(name);
```

STLのコンテナを表4.1に示します。また、STLコンテナの主なメンバ関数を表4.2に示します。

▼表4.1 STLのコンテナ

種類	名前	概要
シーケンスコンテナ	deque	両頭の待ち行列。
	list	一連の値を保存するリスト。双方向のコンテナで、先頭にも要素を挿入可能。
	vector	一連の値を保存する一種の配列。任意の要素にアクセスできる。
連想コンテナ	map	キーと値のペアを保存するコンテナ。要素はソートされ、キーは重複できない。
	multimap	キーと値のペアを保存するコンテナ。要素はソートされ、キーは重複できる。
	set	値をキーとするコンテナ。キーとして使われる値は重複できない。
	multiset	値をキーとするコンテナ。キーとして使われる値は重複できる。
アダプタ	stack	先入れあと出しのスタック。
	queue	先入れ先出し(FIFO)の待ち行列(キュー)。
	priority_queue	優先順位付きのキュー。

▼表4.2 STLコンテナの主なメンバ関数

メンバ関数	機能	実装しているコンテナクラス
empty()	コンテナが空ならtrueを返す。	すべてのクラス
size()	コンテナに保存されている要素数を返す。	
max_size()	コンテナに保存できる要素数の上限を返す。	
swap()	同じ型の他のコンテナと内容を交換する。	
begin()	先頭の要素を指す反復子を返す。	vector、list、deque、set、multiset、map、multimap
end()	最後の要素の次を指す反復子を返す。	
rbegin()	末尾の要素を指す逆反復子を返す。	
rend()	先頭の要素の前を指す逆反復子を返す。	
insert()	コンテナに要素を挿入する。	
erase()	指定した反復子が指す要素をコンテナから削除する。	
clear()	コンテナからすべての要素を削除する。 erase(begin(), end())と同じ。	
front()	先頭要素の参照を返す。	vector、list、deque
back()	末尾の要素の参照を返す。	
push_back()	コンテナの末尾に要素を追加する。	
pop_back()	コンテナから末尾の要素を削除する。	
push_front()	コンテナの先頭に要素を追加する。	list、deque
pop_front()	コンテナから先頭の要素を削除する。	
key_comp()	比較用関数オブジェクトを返す。	set、multiset、map、multimap
value_comp()	比較用関数オブジェクトを返す。	
find()	キーに基づいて要素を探索する。	
lower_bound()	挿入できる最初の位置を探す。	
upper_bound()	挿入できる最後の位置を探す。	
count()	キーと一致する要素の数を返す。	stack、queue、priority_queue
push()	要素を追加する。	
pop()	要素を取り出してコンテナから削除する。	

反復子

反復子（iterator）は、コンテナのそれぞれの要素にアクセスするときに使うものです。反復子という名前が付いているのは、通常、コンテナの各要素に順にアクセスするために繰り返し使われるからです。

たとえば、名前のリストの各要素（すなわち個々の名前）を表示したいときには、次のようにします。

```
list<string>::iterator it = namelist.begin();
for (; it != namelist.end(); it++)
{
   cout << *it << endl;
}
```

反復子はC/C++のポインタに似ています。実際、反復子はある時点でコンテナの中のいずれかの要素を指しているものですし、*（アスタリスク）を付けて実際の値にアクセスする（上の例では*it）ところも、ポインタそっくりです。

アルゴリズム

STLのアルゴリズムは、コンテナの内容を操作するための関数群です。

たとえば、namelistというリストに特定の名前searchnameが含まれているかどうか調べたいときには、find()を使って次のようにします。

```
s = find(namelist.begin(), namelist.end(), searchname);
```

これは、namelistの先頭（namelist.begin()）から最後（namelist.end()の直前にある要素）まで、searchnameと一致する名前を探して返します（begin()は最初の要素を指しますが、end()は最後の要素の後ろの位置を指すということに注意してください）。

結果として返されるのは反復子ですから、一致していれば探した要素に*sでアクセスでき、sがnamelist.end()であったら、最後まで探したものの一致するものがなかったことを示します。

そこで、次のようにすると、リストの中から任意の名前を探してその結果を表示できます。

```
cout << "探す名前は? >";
cin.getline(buff, BUFF_LEN);
list<string>::iterator s;
s = find(namelist.begin(), namelist.end(),
string(buff));
if (s != namelist.end())
   cout << *s << "は登録済みです。" << endl;
else
   cout << buff << "は登録されていません。" << endl;
```

04-01 標準C++ライブラリの概要

STLには豊富なアルゴリズムが用意されています。主なアルゴリズムを表4.3に示します。

▼ 表4.3 STLの主なアルゴリズム

関数	機能
accumulate()	指定した範囲のすべての要素の値を累算した結果を計算する
binary_search()	コンテナの要素に対してバイナリ検索を行う
copy()、copy_backward()	指定した範囲の要素をコピーする
count_if()	特定の条件を満たす要素数を返す
equal()	2つの範囲を比較する
equal_range()	コンテナの順序を変えずに要素を挿入できる最大範囲の位置を返す
fill()、fill_n()	コンテナを特定の値で初期化（再初期化）する
find()、adjacent_find()	要素を検索する
find_first_of()、find_end()	コンテナの中の、サブシーケンスの最初/最後の要素を検索する
find_if()	特定の条件を満足させる最初の要素を検索する
for_each()	コレクションの要素に関数を適用する
generate()、generate_n()	ジェネレータで生成した値でコンテナを初期化（再初期化）する
includes()	2つのソートされたコンテナを比較して、一方の範囲のすべての要素が、もう1つの範囲に含まれている場合にtrueを返す
inplace_merge()	コンテナをマージする
iter_swap()	2つの反復子が指し示す値を交換する
lexicographical_compare()	2つの範囲を辞書順で比較する
lower_bound()	コンテナの値が入る下限を示す反復子を返す
upper_bound()	コンテナの値が入る上限を示す反復子を返す
make_heap()	ヒープを作成する
max()、min()	2つの値の最大値/最小値を返す
max_element()	コンテナの中の最大の要素を示す反復子を返す
min_element()	コンテナの中の最小の要素を示す反復子を返す
merge()	2つのソートされたコンテナをマージして、別のコンテナに入れる
mismatch()	2つのコンテナの一致しない最初の要素を返す
next_permutation()	順序付け関数に従って連続した順列を生成する
nth_element()	比較演算子を使ってコレクションを再配置する
partial_sort()	指定した範囲の値をソートする

partial_sort_copy()	指定した範囲の値をソートして、結果を別のコンテナにコピーする
partition()	条件を満足させる要素を、条件を満足しない要素より前に配置する
pop_heap()	ヒープから値を取り出す
prev_permutation()	順序付け関数に従って連続した順列を生成する
push_heap()	ヒープに値を保存する
random_shuffle()	要素の順番をランダムに変更する
remove()、remove_copy()、remove_copy_if()、remove_if()	要素を削除する。コンテナから要素を実際に削除したい場合はerase()を使う
replace()、replace_copy()、replace_copy_if()、replace_if()	要素を別の値で置き換える。名前にcopyが含まれている関数は、結果を別のコンテナにコピーする
reverse()	要素の順序を反転させる
reverse_copy()	要素の順序を反転させて、結果を別のコンテナにコピーする
rotate()	指定した範囲の値を残りの範囲の値と交換する
rotate_copy()	指定した範囲の値を残りの範囲の値と交換して、結果を別のコンテナにコピーする
search()、search_n()	指定した範囲で値が一致する範囲を検索する
set_difference()、set_symmetric_difference()	要素を操作して、ソートされた差集合を作成する
set_intersection()	要素を操作して、ソートされた交差集合を作成する
set_union()	要素を操作して、ソートされた和集合を作成する
sort()、stable_sort()	要素をソートする
sort_heap()	ヒープをソートされたコレクションに変換する
stable_partition()	条件を満足する要素をすべて満足しない要素より前に配置する
swap()、swap_ranges()	コンテナ全体または指定した範囲の値を交換する
transform()	指定した範囲の値に演算を適用する
unique()	重複している要素を削除する
unique_copy()	重複している要素を削除し、結果を別のコンテナにコピーする

02 標準C++の主な機能とクラス

STLに含まれない標準C++の主な機能とクラスは、次のとおりです。

IOStream

IOStreamは標準ストリームに読み書きするための一連のオブジェクトを提供します。たとえば、入力や出力のための演算子<<と>>、get()、put()、getline()、read()、write()、flush()などの関数を提供しています。

IOStreamは以前のC++ライブラリのIOStreamと同じですが、新しいプログラムでは、以前の<iostream.h>ではなく<iostream>をインクルードして、名前空間stdを使ってください。

例

次の例は、coutとcinを使って変数usrnameに文字列を入力して、そのまま出力する例です。

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char* argv[])
{
   string usrname;
   cout << "Name>";
   cin >> usrname;
   cout << "User=" << usrname << endl;
   return 0;
}
```

実行例

```
Name>Pochi
User=Pochi
```

stringクラス

標準C++には文字列クラスとしてstringクラス（basic_stringクラス）が用意されています。stringクラスはSTLコンテナではありませんが、STLコンテナとほぼ同様なものと考えることができます。

stringクラスでは、原則的に、文字列の保存に必要なメモリが自動的に確保されるので、文字配列に文字列を保存するときのように、バッファの長さについて心配する必要がありません。また、stringオブジェクトは+演算子を使って連結することができます。

stringクラスの使用例を以下に示します。

```
string s1 = "Hello ";
string s2;
s2 = "Gogs";
string s3 = s1 + s2;
// C言語形式の文字列バッファにコピーする
// buffの長さに注意を払う必要がある
strcpy(buff, s3.c_str());
```

complexクラス

complexクラスは、複素数の実数部を表すオブジェクトと、虚数部を表すオブジェクトの2個のオブジェクトを保存するオブジェクトを定義できるクラスです。標準的な操作に必要な、デフォルトのコンストラクタ、デストラクタ、コピーコンストラクタ、代入演算子、浮動小数点型のために定義された算術演算子などが定義されています。

valarrayクラス

valarrayクラスは、複数の要素を含む値の配列を扱います。

最も基本的な代入や四則演算子のほかに、*=、+=、/=、%=、^=、&=などの演算子と、size()、sum()、max()、min()、shift()、そのほか豊富なメンバ関数が用意されています。

numeric_limitsクラス

numeric_limitsクラスは、さまざまな基本データ型の値の範囲や符号などの情報を統一した方法で提供するためのクラスです。たとえば、特定のデータ型に保存できる値の範囲は、従来はCHAR_MIN / CHAR_MAX、SHRT_MIN /SHRT_MAX、FLT_MIN / FLT_MAXなどの定数値で識別しました。

numeric_limitsクラスでは、データ型にかかわらず、最小値はmin()、最大値はmax()で求めることができます。たとえば、文字型の最小値/最大値はnumeric_limits<char>::min() / numeric_limits<char>::max()で、また、浮動小数点型の最小値/最大値はnumeric_limits<float>::min() / numeric_limits<float>::max()で調べることができます。

ロケール

ロケールは、国や地域によって異なる、キャラクタや文字列の評価方法、数字、日付、貨幣の表示形式などに影響を与えます。C言語ではANSI C関数setlocale()を呼び出すとロケールを設定できますが、C++では標準C++のlocaleクラスがあり、キャラクタや文字列の評価、数字、日付、貨幣に関連するメンバ関数が用意されています。

多言語文字セットのサポート

標準C++では、プログラムの国際化をサポートする数多くのクラスが提供されていると同時に、wchar_t型を使ってワイド文字（Unicode）をサポートします。そのため、多言語の文字セットをサポートできます。

メモリ管理機能

標準C++ではプラットフォームに依存しない、一貫した方法でメモリを管理するためのクラスを提供しています。コンテナのためのメモリは、デフォルトのアロケータ（メモリ割り当てのオブジェクト）を使って自動的に割り当てられますが、指定することもできます。

例外処理機能

C/C++のこれまでの例外処理はプラットフォームや処理系に依存していましたが、標準C++では例外処理はexceptionクラスで統一された方法で扱うことができます。

標準C++のクラス

ここでは、標準C++で提供されるクラスをヘッダファイルで整理して示します。たとえば、iostreamは<iostream>をインクルードしたときに使うことができる要素について説明しています。また、これは、<iostream>に含まれる要素を使うときには、<iostream>をインクルードする必要があることを意味します。メンバの解説は「04-03 標準C++のクラスのメンバとアルゴリズム」を参照してください。C言語ライブラリの関数などを利用できるようにするためのクラスについては、「03 C言語リファレンス」の各関数の使い方の説明も参照してください。

関数	機能
<algorithm>	アルゴリズムを利用する（STL）
<bitset>	ビットセットのテンプレートクラスを利用する
<cassert>	C言語ライブラリのアサーションを利用する
<cctype>	文字を分類・変換する
<cerrno>	C言語ライブラリ関数のエラーコードを取得する
<cfloat>	C言語ライブラリ関数の浮動小数点型の定数を利用する
<ciso646>	iso646.hを名前空間stdで利用できるようにする
<climits>	C言語ライブラリの整数型の定数を利用する
<clocale>	C言語ライブラリのlocaleを利用する
<cmath>	C言語ライブラリの数値演算関数を利用する
<complex>	複素数の算術演算をサポートする
<csetjmp>	C言語ライブラリの非ローカルなジャンプを利用する
<csignal>	C言語ライブラリのシグナルを利用する
<cstdarg>	C言語ライブラリの可変個の引数を利用する
<cstddef>	C言語ライブラリの型とマクロを利用する
<cstdio>	C言語ライブラリの入力と出力を利用する
<cstdlib>	C言語ライブラリのstdlibを利用する
<cstring>	C言語ライブラリの文字列操作関数を利用する
<ctime>	C言語ライブラリの時刻・日付の変換関数を利用する
<cwchar>	C言語ライブラリのワイド文字列の操作関数を利用する
<cwctype>	C言語ライブラリのワイド文字の分類関数を利用する
<deque>	シーケンスコンテナdequeを実装する（STL）

04-02　標準C++のクラス

<exception>	例外処理を処理する
<fstream>	外部ファイルへの入出力をサポートする
<functional>	関数オブジェクトを生成する（STL）
<iomanip>	引数が1つの入出力用マニピュレータを定義する
<ios>	入出力ストリームの演算に必要な型と関数を提供する
<iosfwd>	入出力ストリームで使うテンプレートクラスへの前方参照を宣言する
<iostream>	ストリームバッファを使う入出力をサポートする
<istream>	ストリームバッファを使う入力をサポートする
<iterator>	反復子を定義して操作する（STL）
<limits>	数値型の範囲などを取得する
<list>	シーケンスコンテナlistを実装する（STL）
<locale>	ロケールの影響を受ける関数をサポートする
<map>	連想コンテナmapを実装する（STL）
<memory>	オブジェクトにメモリを割り当てたり解放する（STL）
<multimap>	連想コンテナmultimapを実装する（STL）
<multiset>	連想コンテナmultisetを実装する（STL）
<numeric>	数値計算に役立つテンプレート関数を定義する（STL）
<ostream>	ストリームバッファを使う出力のサポート
<priority_queue>	アダプタpriority_queueを実装する（STL）
<queue>	アダプタqueueを実装する（STL）
<set>	連想コンテナsetを実装する（STL）
<sstream>	文字列コンテナへの入出力をサポートする
<stack>	アダプタstackを実装する（STL）
<stdexcept>	例外を報告する
<streambuf>	入出力ストリームの演算をバッファリングする
<string>	文字列に関する演算子と関数を提供する
<strstream>	メモリ上の文字配列に対するストリーム入出力をサポートする
<utility>	オブジェクトのペアの比較を可能にする（STL）
<valarray>	値の1次元配列をサポートする
<vector>	シーケンスコンテナvectorを実装する（STL）

01 アルゴリズムを利用する（STL） <algorithm>

書式　#include <algorithm>

機能

コンテナに対して提供するさまざまなアルゴリズムが定義されています。STLには豊富なアルゴリズムが用意されています。主なアルゴリズムは、「表4.3 STLの主なアルゴリズム」を参照してください。

02 ビットセットのテンプレートクラスを利用する <bitset>

書式　#include <bitset>

機能

Nビットのオブジェクトを複数まとめて保存するためのクラスです。

例

次の例は、最初にサイズが16ビットのビットセットを作り、その偶数ビットを１にします。それからビットを反転して、最後にリセットします。

```
#include <iostream>
#include <bitset>
#include <string>

using namespace std;

int main(int argc, char* argv[])
{
   // サイズが16ビットのビットセットを作る
   bitset< 16 > bs;

   // 偶数ビットを１にする
   for (int i=0; i<8; i++)
      bs.set(i*2);
   cout << bs.to_string<char, char_traits<char>, allocator<char> >() << endl;

   bs.flip(); // ビットを反転する
   cout << bs.to_string<char, char_traits<char>, allocator<char> >() << endl;

   bs.reset(); // リセットする
   cout << bs.to_string<char, char_traits<char>, allocator<char> >() << endl;

   return 0;
}
```

● 実行例

```
0101010101010101
1010101010101010
0000000000000000
```

03 C言語ライブラリのアサーションを利用する <cassert>

書式	#include <cassert>

● 機能

C言語ライブラリの<assert.h>を名前空間stdでインクルードするときに使います。C言語の関数assert()は、引数の式を評価した結果がfalseであると、診断メッセージを表示してプログラムを停止します。

04 文字を分類・変換する <cctype>

書式	#include <cctype>

● 機能

C言語ライブラリの<ctype.h>を名前空間stdでインクルードするときに使います。C言語のctype.hに含まれる関数は、引数の文字の種類を判別したり文字を変換します。

● 例

次の例は、cctypeを使ってC言語の関数toupper()を使う例です。

```
#include <iostream>
#include <string>
#include <cctype> // #include <ctype.h>と同じ効果

using namespace std;

int main(int argc, char* argv[])
{
   string s = "pochi";
   cout << s << endl;
   s.at(0) = toupper(s[0]);
   cout << s << endl;

   return 0;
}
```

● **実行例**

```
pochi
Pochi
```

● **備考**

そのほかのC言語ライブラリを利用するヘッダファイル（clocale、cmath、cstdlibなど）の使い方も同様です。

● **参照**→<locale>

05 C言語ライブラリ関数のエラーコードを取得する <cerrno>

書式	#include <cerrno>

● **機能**… C言語ライブラリの<errno.h>を名前空間stdでインクルードするときに使います。

● **参照**→errno

06 C言語ライブラリ関数の浮動小数点型の定数を利用する <cfloat>

書式	#include <cfloat>

● **機能**

C言語ライブラリの<float.h>を名前空間stdでインクルードするときに使います。C言語のfloat.hには浮動小数点演算で使う定数が宣言されています。

07 iso646.hを名前空間stdで利用できるようにする <ciso646>

書式	#include <ciso646>

● **機能**

C言語ライブラリの<iso646.h>を名前空間stdでインクルードするときに使います。

iso646.hには次のような定数が宣言されています。

▼表 iso646.hの定数

定数	値
and	&&
and_eq	&=
bitand	&
bitor	|
compl	~
not	!
not_eq	!=
or	||
or_eq	|=
xor	^
xor_eq	^=

08 C言語ライブラリの整数型の定数を利用する <climits>

書式 #include <climits>

● 機能

C言語ライブラリのlimits.hを名前空間stdでインクルードするときに使います。C言語のlimits.hには値の最大値や最小値が宣言されています。C++では可能な限りnumeric_limitsクラスを使ってください。

09 C言語ライブラリのlocaleを利用する <clocale>

書式 #include <clocale>

● 機能

C言語ライブラリのlocale.hを名前空間stdでインクルードするときに使います。locale.hにはロケールのカテゴリ定数やロケールで使う構造体が宣言されています。C++では可能な限り<locale>を使ってください。

10 C言語ライブラリの数値演算関数を利用する <cmath>

書式	#include <cmath>

● 機能

C言語ライブラリのmath.hを名前空間stdでインクルードするときに使います。

11 複素数の算術演算をサポートする <complex>

書式	#include <complex>

● 機能

複素数の実数部を表すオブジェクトと、虚数部を表すオブジェクトの2個のオブジェクトを保存するオブジェクトを定義できるクラスです。標準的な操作に必要な、デフォルトのコンストラクタ、デストラクタ、コピーコンストラクタ、代入演算子、浮動小数点型のために定義された算術演算子などが宣言されています。

● 例

次の例は、complexの使用例です。

```
#include <iostream>
#include <complex>

using namespace std;

int main(int argc, char* argv[])
{
   complex< double > c(3.14);
   cout << c.real() << endl;
   c *= 2;
   cout << c.real() << endl;
   return 0;
}
```

● 実行例

```
3.14
6.28
```

12 C言語ライブラリの非ローカルなジャンプを利用する <csetjmp>

書式	#include <csetjmp>

機能

C言語ライブラリのsetjmp.hを名前空間stdでインクルードするときに使います。

setjmp.hにはsetjmp()と関連する構造体などが宣言されています。

13 C言語ライブラリのシグナルを利用する <csignal>

書式	#include <csignal>

機能

C言語ライブラリのsignal.hを名前空間stdでインクルードするときに使います。signal.hにはシグナル関数signal()と関連する関数raise()や定数などが宣言されています。

14 C言語ライブラリの可変個の引数を利用する <cstdarg>

書式	#include <cstdarg>

機能

C言語ライブラリのstdarg.hを名前空間stdでインクルードするときに使います。stdarg.hには関数の可変個の引数を実現するための関数や構造体が宣言されています。

15 C言語ライブラリの型とマクロを利用する <cstddef>

書式	#include <cstddef>

機能

C言語ライブラリのstddef.hを名前空間stdでインクルードするときに使います。

stddef.hにはいくつかの定数や型が宣言されています。

16 C言語ライブラリの入力と出力を利用する <cstdio>

書式	#include <cstdio>

機能

C言語ライブラリのstdio.hを名前空間stdでインクルードするときに使います。

stdio.hには標準入出力のための関数、構造体や定数が宣言されています。また、stdin、stdout、stderrも定義されています。

stdinやstdoutと、cinやcoutを混在させて使うことは、好ましくありません。

17 C言語ライブラリのstdlibを利用する <cstdlib>

書式　#include <cstdlib>

機能

C言語ライブラリのstdlib.hを名前空間stdでインクルードするときに使います。

stdlib.hにはC言語のプログラミングでよく使われるatol()やstrtod()のような変換関数、定数、構造体などが宣言されています。

18 C言語ライブラリの文字列操作関数を利用する <cstring>

書式　#include <cstring>

機能

C言語ライブラリのstring.hを名前空間stdでインクルードするときに使います。string.hにはC言語ライブラリの文字列関数やメモリ関数、構造体などが宣言されています。

C++のstringをC言語の文字列として扱いたいときには、c_str()を使います。

プログラミング豆知識

マクロの副作用

#defineを使ってマクロを定義するときには、誤って解釈されないように十分に注意を払う必要があります。たとえば、次のマクロMASK(x)とmask(x)は同じように見えますが、MASK(x)の方は使い方によっては意図しない結果になってしまいます。

```
#include <stdio.h>
#define MASK(x) x & 0x10
#define mask(x) (x & 0x10)

int main(int argc, char* argv[])
{
   int n = 0x11;
   printf("%d¥n", 0x10 - MASK(n)); // 結果は16
   printf("%d¥n", 0x10 - mask(n)); // 結果は0
}
```

このようなマクロの副作用を防ぐために、上の例のmask()と同様に、マクロとして定義する式はかっこ()で囲む習慣を付けると良いでしょう。

19 C言語ライブラリの時刻・日付の変換関数を利用する <ctime>

書式 #include <ctime>

● 機能

C言語ライブラリのtime.hを名前空間stdでインクルードするときに使います。time.hには時刻や日付の変換関数や構造体などが宣言されています。

20 C言語ライブラリのワイド文字列の操作関数を利用する <cwchar>

書式 #include <cwchar>

● 機能

C言語ライブラリのwchar.hを名前空間stdでインクルードするときに使います。wchar.hにはワイド文字を調べたり文字列の操作を行うための関数や構造体などが宣言されています。

21 C言語ライブラリのワイド文字の分類関数を利用する <cwctype>

書式 #include <cwctype>

● 機能

C言語ライブラリのwctype.hを名前空間stdでインクルードするときに使います。wctype.hにはワイド文字を調べたり変換するための関数や構造体などが宣言されています。

22 シーケンスコンテナdequeを実装する（STL） <deque>

書式 #include <deque>

● 機能

dequeはシーケンスコンテナの一種で、両頭の待ち行列のコンテナです。

● 例

次の例は、intのdequeを作成して値を保存する例です。

```
#include <iostream>
#include <deque>

using namespace std;

int main(int argc, char* argv[])
{
   deque< int > dq;
   dq.push_front(1);
   dq.push_front(2);
   dq.push_back(3);
   dq.push_back(4);
   cout << dq.size() << ":";
   int i;
   for (i=0; i<dq.size(); i++)
      cout << dq[i] << " ";
   cout << endl;
   dq.pop_front();
   cout << dq.size() << ":";
   for (i=0; i<dq.size(); i++)
      cout << dq[i] << " ";
   cout << endl;
   return 0;
}
```

● 実行例

```
4:2 1 3 4
3:1 3 4
```

● 備考

その他のシーケンスコンテナの使い方も同様です。

23 例外処理を処理する <exception>

書式	#include <exception>

● 機能

標準C++ライブラリと特定の式からスローされるあらゆる例外の基本クラスです。

24 外部ファイルへの入出力をサポートする <fstream>

書式	#include <fstream>

●機能

外部ファイルに保存されたデータの入出力操作をサポートするためのクラスです。このクラスは、iostreamから派生したクラスです。

25 関数オブジェクトを生成する（STL） <functional>

書式	#include <functional>

●機能

関数オブジェクトを生成するときに使います。関数オブジェクトは，定義された関数呼び出し演算子（( ) 演算子）を持つオブジェクトで、標準ライブラリの汎用アルゴリズムの引数として指定できるので、アルゴリズムを効果的に利用するときに使います。

●例

次の例は、最初の引数が2番目の引数より小さいか等しければtrueを返す二項関数オブジェクトless_equalを使って値を小さい順に並べ替えるプログラムの例です。

```
#include <algorithm>
#include <vector>
#include <iostream>
#include <functional>

using namespace std;

int main(void)
{
   typedef vector<int>::iterator iterator;

   int d[] = {11, 8, 6, 4, 7};
   vector<int> v(d, d + 5);

   // コンテナの内容を出力する
   ostream_iterator<int,char> out(cout, ",");
   cout << "before:" << endl;
   copy(v.begin(), v.end(), out);
   cout << endl;
```

```
    sort(v.begin(), v.end(), less_equal<int>());

    // 結果を出力する
    cout << "after:" << endl;
    copy(v.begin(), v.end(), out);
    cout << endl;

    return 0;
}
```

26 引数が1つの入出力用マニピュレータを定義する <iomanip>

書式 #include <iomanip>

機能

引数が1つの入出力用マニピュレータを定義します。たとえば、数値を16進数で出力したいときは、<iomanip>のsetbase(16)を、数値を8進数で出力したいときはsetbase(8)を使うことができます。

例

次の例は、数値を16進数で表示するプログラムの例です。

```
#include <iostream>
#include <iomanip>

using namespace std;

int main(int argc, char *argv[])
{
    int v = 30;

    // 数値を16進数で出力する
    cout << setbase(16) << v << endl;

    return 0;
}
```

27 入出力ストリームの演算に必要な型と関数を提供する <ios>

書式	#include <ios>

● 機能

入出力ストリームの演算に必要な型と関数を提供します。関数のほとんどはマニピュレータで、入力ストリームから抽出したり、出力ストリームに挿入することができます。

入出力ストリームを利用するときに使うフラグ(ios:flags)を示します。

▼表 入出力ストリームのフラグ

フラグ	機能
ios::skipws	入力の中の空白文字をスキップする。
ios::left	値を左詰めにする。右側はパディング文字で埋める。
ios::right	値を右詰めにする。左側はパディング文字で埋める（デフォルト）。
ios::internal	符号または基数表示と値の間にパディング文字を追加する。
ios::dec	数値を10進数にする（デフォルト）。
ios::oct	数値を8進数にする。
ios::hex	数値を16進数にする。
ios::showbase	基数を表示する。
ios::noshowbase	基数を表示しない。
ios::showpoint	浮動小数点数のときに小数点と後続する0を表示する。
ios::noshowpoint	浮動小数点数のときに小数点と後続する0を表示しない。
ios::uppercase	16進数のAからFおよび指数表現のEを大文字で表示する。
ios::showpos	正の値のとき符号（+）を表示する。
ios::scientific	浮動小数点数を指数表記で表示する。
ios::fixed	浮動小数点数を固定小数点表記で表示する。
ios::unitbuf	挿入するごとにストリーム（デフォルトではcerr）をフラッシュする。
ios::stdio	挿入するごとにstdoutとstrerrをフラッシュする。

28 入出力ストリームで使うテンプレートクラスへの前方参照を宣言する <iosfwd>

書式	#include <iosfwd>

● 機能

入出力ストリームで使う一連のテンプレートクラスへの前方参照を宣言します。ほかの標準ヘッダで定義されているテンプレートクラスの定義でなく宣言だけを使うときに、このヘッダを明示的にインクルードします。

29 ストリームバッファを使う入出力をサポートする <iostream>

書式 #include <iostream>

●機能

ストリームバッファで制御されるシーケンスからの入出力をサポートします。書式付き入出力と書式なし入出力をサポートします。名前空間stdに定義されているcin、cout、cerr、clog、wcin、wcout、wcerr、wclogなどの基本的なストリーム入出力オブジェクトを使うときに必要です。

●例

次の例は、標準出力ストリーム（cout）と標準入力ストリーム（cin）を使って変数sに文字列を入力し、その長さを文字列と共に出力する例です。文字列の長さが64を超えた場合は標準エラー出力（cerr）にメッセージ「文字列の長さが長すぎ。」を出力します。

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char* argv[])
{
   string s;
   cout << "文字列>";
   cin >> s;
   if (s.length() > 64)
      cerr << "文字列の長さが長すぎ。" << endl;
   else
      cout << s << "の長さ=" << s.length() << endl;
   return 0;
}
```

30 ストリームバッファを使う入力をサポートする <istream>

書式 #include <istream>

●機能

ストリームバッファで制御されるシーケンスからの入力をサポートします。書式付き入力と書式なし入力をサポートします。

31 反復子を定義して操作する（STL） <iterator>

書式 #include <iterator>

●機能

反復子（iterator）を定義して利用するときに必要です。

32 数値型の範囲などを取得する <limits>

書式 #include <limits>

●機能

テンプレートクラスnumeric_limitsを利用するときに必要です。

33 シーケンスコンテナlistを実装する（STL） <list>

書式 #include <list>

●機能

listはシーケンスコンテナの一種で、一連の値を保存するリストのコンテナです。これは双方向のコンテナで、両方向性反復子をサポートし、先頭にも要素を挿入できます。

●**例**… find()の例参照。

34 ロケールの影響を受ける関数をサポートする <locale>

書式 #include <locale>

●機能

ロケールの影響を受ける関数をサポートします。次のような関数があります。

codecvt()	ctype<char>()	iscntrl()	isspace()
codecvt_base()	ctype_base()	isdigit()	isupper()
codecvt_byname()	ctype_byname()	isgraph()	isxdigit()
collate()	has_facet()	islower()	locale()
collate_byname()	isalnum()	isprint()	messages()
ctype()	isalpha()	ispunct()	messages_base()

messages_byname()	moneypunct_byname()	time_base()	tolower()
money_base()	num_get()	time_get()	toupper()
money_get()	num_put()	time_get_byname()	use_facet()
money_put()	numpunct()	time_put()	
moneypunct()	numpunct_byname()	time_put_byname()	

35 連想コンテナmapを実装する（STL）<map>

書式 #include <map>

●機能

mapは連想コンテナの一種で、キーと値のペアを保存します。要素はソートされ、キーは重複できません。

●例

次の例は、mapを使って整数（int）と文字型（char）の値のペアを保存する例です。

```
#include <iostream>
#include <map>

using namespace std;

int main(int argc, char* argv[])
{
   typedef map< int, char > mis;
   mis aMap;

   aMap.insert( mis::value_type( 5, 'e' ) );
   aMap.insert( mis::value_type( 3, 'c' ) );
   aMap.insert( mis::value_type( 1, 'a' ) );
   aMap.insert( mis::value_type( 26, 'z' ) );
   aMap.insert( mis::value_type( 2, 'b' ) );
   cout << "サイズ=" << aMap.size() << endl;
   mis::const_iterator iter;
   for (iter = aMap.begin(); iter != aMap.end(); ++ iter)
   {
      cout << (*iter).first << " -> ";
      cout << (*iter).second << endl;
   }
   return 0;
}
```

● 実行例

```
サイズ=5
1 -> a
2 -> b
3 -> c
5 -> e
26 -> z
```

36 オブジェクトにメモリを割り当てたり解放する（STL） <memory>

書式	#include <memory>

● 機能

オブジェクトの割り当てと解放を行うクラス、演算子、テンプレートを定義するときに使います。

37 連想コンテナmultimapを実装する（STL） <multimap>

書式	#include <multimap>

● 機能

multimapは連想コンテナの一種で、キーと値のペアを保存します。要素はソートされ、キーを重複できます。

● 参照→<map>

38 連想コンテナmultisetを実装する（STL） <multiset>

書式	#include <multiset>

● 機能

multisetは連想コンテナの一種で、値をキーとするコンテナです。キーとして使われる値を重複できます。

● 参照→<set>

39 数値計算に役立つテンプレート関数を定義する（STL）<numeric>

書式	#include <numeric>

● **機能**

数値計算に役立つテンプレート関数を定義するときに使います。

40 ストリームバッファを使う出力のサポート <ostream>

書式	#include <ostream>

● **機能**

ストリームバッファで制御されるシーケンスへの出力をサポートします。書式付き出力と書式なし出力をサポートします。

41 アダプタpriority_queueを実装する（STL）<priority_queue>

書式	#include <priority_queue>

● **機能**

priority_queueはアダプタの一種で、優先順位付きのキューコンテナです。

42 アダプタqueueを実装する（STL）<queue>

書式	#include <queue>

● **機能**

queueはアダプタの一種で、先入れ先出し（FIFO）の待ち行列（キュー）のコンテナです。

43 連想コンテナsetを実装する（STL）<set>

書式	#include <set>

● **機能**

setは連想コンテナの一種で、値をキーとするコンテナです。キーとして使われる値は重複できません。

44 文字列コンテナへの入出力をサポートする <sstream>

書式	#include <sstream>

●機能

文字列コンテナへのシーケンスへの入出力をサポートします。

45 アダプタstackを実装する（STL） <stack>

書式	#include <stack>

●機能

stackはアダプタの一種で、先入れあと出しのスタックのコンテナです。stackはアダプタなので、データを保存するために連接コンテナ（vector、list、deque）のいずれかを使います。

●例

次の例は、vectorコンテナを使ったstackに整数を保存する例です。

```
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

int main(int argc, char* argv[])
{
   stack< int , vector< int > > stk;
   stk.push(1);
   stk.push(2);
   stk.push(3);
   stk.push(4);
   stk.push(5);

   while (!stk.empty())
   {
      cout << stk.top() << " ";
      stk.pop();
   }
   cout << endl;

   return 0;
}
```

● 実行例

```
5 4 3 2 1
```

● 例

次の例は、stackに整数を保存して取り出す例です。この例では連接コンテナを指定していないので、デフォルトのdequeが使われます。

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <stack>

using namespace std;

int main(int argc, char *argv[])
{
   stack<int> stk;

   srand( (unsigned)time( NULL ) );

   // コンテナに値をプッシュする
   for(int i = 0 ; i < 5; ++i){
        stk.push( rand() % 100 );
   }

   // コンテナから値をポップする
   while( !stk.empty() ){
        cout << stk.top() << endl;
        stk.pop();
   }

   return 0;
}
```

46 例外を報告する <stdexcept>

書式	#include <stdexcept>

● 機能

プログラムの実行時に検出されるエラーを報告するためにスローされるすべての例外を扱います。

47 入出力ストリームの演算をバッファリングする <streambuf>

書式	#include <streambuf>

●機能

入出力ストリームクラスの演算に基本的なテンプレートクラスを利用できるようにします。

このヘッダはほかの入出力ストリームヘッダによって自動的にインクルードされるので、通常、直接インクルードする必要はありません。

48 文字列に関する演算子と関数を提供する <string>

書式	#include <string>

●機能

文字列コンテナに関する基本的な演算子と関数を提供します。

49 メモリ上の文字配列に対するストリーム入出力をサポートする <strstream>

書式	#include <strstream>

●機能

メモリ上のchar配列に対するストリーム入出力を行うときに使います。このオブジェクトはC言語の文字列と相互に変換できます。このクラスは、iostreamから派生したクラスです。

50 オブジェクトのペアの比較を可能にする（STL） <utility>

書式	#include <utility>

●機能

基本演算子operator==とoperator<を定義することで、同じ型のペアのオブジェクトに対する4種類のテンプレート演算子（operator!=、operator<=、operator>、operator>=）の完全な順序付けを定義できるようにします。

51 値の1次元配列をサポートする <valarray>

書式	#include <valarray>

機能

値の1次元配列にデータを保存したり操作したりするために使います。配列の要素は0から始まるインデックスで識別できます。valarrayは効率がよい代わりに、一定の範囲の型以外には使えません。

関連するクラスとして、slice_array、gslice_array、mask_array、indirect_arrayがあります。

52 シーケンスコンテナvectorを実装する（STL） <vector>

書式	#include <vector>

機能

vectorはシーケンスコンテナの一種で、一連の値を保存する一種の配列と考えることができます。任意の要素にアクセスできるコンテナです。

例

accumulate()、adjacent_find()の例、fill()などの例参照。

標準C++のクラスのメンバとアルゴリズム

ここでは標準C++の各クラスに含まれる関数、マニピュレータやアルゴリズムのうち、よく使われるものを、アルファベット順に概説します。

名前またはシンボル	機能
<<	ストリームへデータを出力する
>>	ストリームからデータを読み込む
accumulate()	指定した範囲のすべての要素の値を累算した結果を計算する
adjacent_find()	シーケンスの中の等しいペアを検索する
back()	シーケンスの最後の要素を指す参照を返す
begin()	先頭の位置を指す反復子を返す
binary_search()	コンテナの要素に対してバイナリ検索を行う
c_str()	stringオブジェクトをC言語形式の文字列定数に変換する
close()	オブジェクトを閉じる
copy()	オブジェクトをコピーする
copy_backward()	指定した範囲の要素をコピーする
count()	範囲または全体の要素の数を返す
count_if()	特定の条件を満たす要素数を返す
dec	10進数に変換する
eatwhite()	先行する空白文字を取り出す
empty()	コンテナが空であるかどうか調べる
end()	最後の位置を指す反復子を返す
endl	改行文字を出力ストリームに出力する
ends	NULLバイトを出力ストリームに出力する
equal()	2つの範囲を比較する
equal_range()	コンテナの順序を変えずに要素を挿入できる最大範囲の位置を返す
erase()	シーケンスから要素を消去する
fill()	コンテナを特定の値で初期化（再初期化）する
fill_n()	コンテナを特定の値で初期化（再初期化）する
find()	コンテナの要素を検索する
find_end()	コンテナの中のサブシーケンスの最後の要素を検索する
find_first_of()	コンテナの中のサブシーケンスの最初の要素を検索する
find_if()	特定の条件を満足させる最初の要素を検索する

flush	出力ストリームをフラッシュする
for_each()	コレクションの要素に関数を適用する
front()	シーケンスの最初の要素を指す参照を返す
fstream()	ファイルストリームを作成する
gcount()	直前の入力関数で抽出された文字数を返す
generate()	ジェネレータで生成した値でコンテナを初期化（再初期化）する
generate_n()	ジェネレータで生成した値でコンテナを初期化（再初期化）する
get()	データを取得する
getline()	ストリームから行単位で文字列を入力する
good()	入出力ストリームのエラー状態を返す
hex	16進数に変換する
ignore()	ストリームから文字を取り出して廃棄する
includes()	一方の範囲のすべての要素が他の範囲に含まれているか調べる
inplace_merge()	コンテナをマージする
insert()	シーケンスに要素を挿入する
ipfx()	ストリーム入力の準備をする
isfx()	ストリームからの入力終了の処理をする
iter_swap()	2つの反復子が指し示す値を交換する
key_comp()	要素の順序を決定する比較関数オブジェクトを返す
lexicographical_compare()	2つの範囲を辞書順で比較する
lower_bound()	コンテナの値が入る下限を示す反復子を返す
make_heap()	ヒープを作成する
max()	最大値を返す
max_element()	コンテナの中の最大の要素を示す反復子を返す
max_size()	要素数の上限を返す
merge()	2つのソートされたコンテナをマージする
min()	最小値を返す
min_element()	コンテナの中の最小の要素を示す反復子を返す
mismatch()	2つのコンテナの一致しない最初の要素を返す
next_permutation()	順序付け関数に従って連続した順列を生成する
nth_element()	比較演算子を使ってコレクションを再配置する
oct	8進数に変換する

04-03 標準C++のクラスのメンバとアルゴリズム

open()	オブジェクトを開く
operator <<	ストリームへデータを出力する
operator >>	ストリームからデータを読み込む
partial_sort()	指定した範囲の値をソートする
partial_sort_copy()	指定した範囲の値をソートして別のコンテナに保存する
partition()	条件を満足させる要素を、条件を満足しない要素より前に配置する
peek()	ストリームから文字を取り出さずに文字を返す
pop()	シーケンスの最後の要素を除去する
pop_back()	シーケンスの最後の要素を除去する
pop_front()	シーケンスの最初の要素を除去する
pop_heap()	ヒープから要素を取り出す
prev_permutation()	順序付け関数に従って連続した順列を生成する
push()	シーケンスの最後に要素を挿入する
push_back()	シーケンスの最後に要素を挿入する
push_front()	シーケンスの先頭に要素を挿入する
push_heap()	ヒープに値を保存する
putback()	ストリームに文字を戻す
random_shuffle()	要素の順番をランダムに変更する
rbegin()	逆シーケンスの先頭の位置を指す反復子を返す
read()	ストリームからデータを取り出す
readsome()	ストリームから文字を取り出す
remove()	要素を削除する
remove_copy()	要素を削除する
remove_copy_if()	要素を削除する
remove_if()	要素を削除する
rend()	逆シーケンスの終端を指す反復子を返す
replace()	要素を別の値で置き換える
replace_copy()	要素を置き換え結果を別のコンテナに保存する
replace_copy_if()	要素を置き換え結果を別のコンテナに保存する
replace_if()	要素を別の値で置き換える
resetiosflags	フラグをリセットする
resize()	コンテナのサイズを再設定する
reverse()	要素の順序を反転させる

reverse_copy()	要素の順序を反転させ結果を別のコンテナに保存する
rotate()	指定した範囲の値を残りの範囲の値と交換する
rotate_copy()	指定した範囲の値を残りの範囲の値と交換し別のコンテナに保存する
search()	指定した範囲で値が一致する範囲を検索する
search_n()	指定した範囲で値が一致する範囲を検索する
seekg()	ストリームの入力ポインタを変更する
set_difference()	ソートされた差集合を作成する
set_intersection()	ソートされた交差集合を作成する
set_symmetric_difference()	ソートされた対称差集合を作成する
set_union()	ソートされた和集合を作成する
setbase()	ストリームへの入出力の際の基数を設定する
setfill()	ストリームへのパディング文字を設定する
setiosflags()	フラグを設定する
setprecision()	ストリームへの入出力の小数点以下の精度を設定する
setw	ストリームからの入出力のフィールド幅を設定する
size()	シーケンスのサイズを返す
sort()	要素をソートする
sort_heap()	ヒープをソートされたコレクションに変換する
stable_partition()	条件を満足する要素をすべて満足しない要素より前に配置する
stable_sort()	要素をソートする
swap()	コンテナ全体の値を交換する
swap_ranges()	コンテナの指定した範囲の値を交換する
sync()	ストリームバッファと外部から入力される文字との同期をとる
tellg()	ストリームの入力ポインタの値を取得する
tolower()	文字を小文字に変換する
toupper()	文字を大文字に変換する
transform()	指定した範囲の値に演算を適用する
unique()	重複している要素を削除する
unique_copy()	重複している要素を削除し、別のコンテナに保存する
upper_bound()	コンテナの値が入る上限を示す反復子を返す
value_comp()	要素の順序を決定する関数オブジェクトを返す
write()	バッファからストリームにバイト列を出力する
ws	先行する空白文字を取り除く

01 ストリームへデータを出力する
<<

● 参照→operator <<

02 ストリームからデータを読み込む
>>

● 参照→operator >>

03 指定した範囲のすべての要素の値を累算した結果を計算する
accumulate()

ヘッダ	numeric
書式	① template<class iiter, class T> T accumulate(iiter *first*, iiter *last*, T *val*); ② template<class iiter, class T, class pred> T accumulate (iiter *first*, iiter *last*, T *val*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *val*………アキュムレーターを初期化する値。 *pr* ………各要素に適用するバイナリ操作。

● **解説**

このアルゴリズムは、指定した範囲のすべての要素の値を累算した結果を計算します。

● **例**

次の例は、整数値1～10をvectorに保存し、アルゴリズムaccumulate()を使って合計を求め、出力する例です。

```
#include <numeric>
#include <vector>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   typedef vector<int>::iterator iterator;
   int d[10] = {1,2,3,4,5,6,7,8,9,10};
```

```
    vector<int> v(d, d+10);

    int sum = accumulate(v.begin(), v.end(), 0);
    cout << "1～10の合計= " << sum << endl;

    return 0;
}
```

04 シーケンスの中の等しいペアを検索する adjacent_find()

ヘッダ	algorithm
書式	① template<class iter> iter adjacent_find(iter *first*, iter *last*); ② template<class iter, class pred> iter adjacent_find(iter *first*, iter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………各要素に適用するバイナリ述語。

●解説

このアルゴリズムは、シーケンス内にある隣接した要素のペアで、最初の等しいペアを見つけます。

●例

次の例は、12個の整数値をvectorに保存したあとでアルゴリズムsort()を使ってソートし、さらにアルゴリズムadjacent_find()を使って同じ値で最小の隣接するペアの値を探します。

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    typedef vector<int>::iterator iterator;
    int d[] = {1,12,9,9,4,6,5,6,7,8,5,7};
```

```
    vector<int> v(d, d+12);
    // 要素を並べ替える
    sort(v.begin(), v.end());
    // 同じ値のペアを探す
    iterator it = adjacent_find(v.begin(),v.end());
    cout << "同じ値の最小のペア= " << *it << endl;
    return 0;
}
```

05 シーケンスの最後の要素を指す参照を返す back()

書式	① value_type& QSContainer::back(); ② const value_type& back() const;

● 解説

空ではないシーケンスの最後の要素を指す参照を返します。

06 先頭の位置を指す反復子を返す begin()

書式	① const_iterator Container::begin() const; ② iterator Container::begin();

● 解説

シーケンスの最初の要素または空であるシーケンスの直後の位置を指す反復子を返します。

● 例

accumulate()やadjacent_find()の例参照。

07 コンテナの要素に対してバイナリ検索を行う binary_search()

ヘッダ	algorithm
書式	① template<class iter, class T> bool binary_search(iter *first*, iter *last*, const T& *val*); ② template<class iter, class T, class pred> bool binary_search(iter *first*, iter *last*, const T& *val*, Compare *comp*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *val*………探す値。 *comp* …比較述語。

●解説

このアルゴリズムは、コンテナの要素に対してバイナリ検索を行います。

08 stringオブジェクトをC言語形式の文字列定数に変換する c_str()

書式	const E *basic_string::c_str() const;

●解説

stringオブジェクトをC言語形式の文字列に変換します。変換された文字列は定数であるので、そのままでは変更できません。変更したいときには、別のバッファに保存してから変更する必要があります。

●例

次の例は、C言語の関数を使うためにstringオブジェクトをC言語形式の文字列に変換する例です。

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char *argv[])
{
    string s = "0123456789abcdef";

    char buff[128];
```

```
    // C言語形式の文字列バッファに保存する
    strncpy(buff, s.c_str(), 5);

    // 文字列を追加する
    strcat(buff, " and ABC");

    // 標準出力に出力する
    cout << buff << endl;

    return 0;
}
```

09 オブジェクトを閉じる close()

書式

① void fstream::close();
② basic_filebuf *basic_filebuf::close();
③ void basic_fstream::close();
④ void basic_ifstream::close();
⑤ void basic_ofstream::close();
⑥ catalog messages::close(catalog cat) const;

● 解説

ディスク上のファイルやメッセージカタログなどを閉じます。

● 参照→open()

● 例

write()の例参照

10 オブジェクトをコピーする copy()

書式	① template<class iiter, class oiter> oiter copy(iiter *first*, iiter *last*, oiter *x*); ② size_type basic_string::copy(E **s*, size_type *n*, size_type *pos* = 0) const; ③ static E *char_traits::copy(E **xx*, const E **yy*, size_t *n*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………出力反復子。 *s* ………コピーする文字列。 *n* ………コピーする数。 *pos* ……コピーする先頭の位置。 *xx* ………コピーする元。 *yy* ………コピー先。

● 解説

書式①のアルゴリズムは指定した範囲の要素をコピーします。書式②と③は、それぞれ文字列や文字配列をコピーします。

● 例

次の例は、vectorの内容をコピーするプログラムの例です。

```
#include <iostream>
#include <vector>
#include <iterator>

using namespace std;

typedef ostream_iterator <int, char, char_traits <char> > osit;

int main(int argc, char *argv[])
{
   vector < int > v;

   osit ovstream(cout," ");

   for (int i=0; i<10; i++)
      v.push_back(i);

   copy(v.begin(), v.end(), ovstream);

   cout << endl;

   return 0;
}
```

11 指定した範囲の要素をコピーする
copy_backward()

ヘッダ	algorithm
書式	template<class biter1, class biter2> biter2 copy_backward (biter1 *first*, biter1 *last*, biter2 *x*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………出力反復子。

● 解説

このアルゴリズムは、指定した範囲の要素をシーケンスの末尾の要素から始めて先頭へとコピーします。要素自体が後ろ向きにコピーされるのではなく、移動の順序がcopyと逆になり、移動されたあとの値の相対的な位置はコピー元の位置と同じです。

12 範囲または全体の要素の数を返す
count()

ヘッダー	algorithm
書式	① template<class InIt, class T> size_t count(InIt *first*, InIt *last*, const T& *val*); ② size_type AssoCont::count(const Key& *key*) const; ③ size_t bitset::count() const;
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *val*………一致する値。 *key* ……一致するキー。

● 解説

書式①の関数は、指定した範囲にある要素の数を返します。

書式②の関数は、連想コンテナの範囲（lower_bound(key), upper_bound(key)）にある要素の数を返します。

書式③の関数は、ビットシーケンスのセットされたビットの個数を返します。

●例

次の例はvectorに20個のランダムな数を保存し、2である要素数を調べる例です。

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
   vector<int> v;

   srand( (unsigned)time( NULL ) );

   // コンテナに値をプッシュする
   for(int i = 0 ; i < 20; ++i)
        v.push_back( rand() % 10 );

   for (int i=0; i< (int)v.size(); i++)
     cout << v[i] << ",";
   cout << endl;

   int n = count(v.begin(), v.end(), 2);

   cout << "2は" << n << "個" << endl;

   return 0;
}
```

●実行例

```
9,8,9,4,6,0,0,9,1,6,9,8,7,4,2,9,2,2,5,1,
2は3個
```

13 特定の条件を満たす要素数を返す count_if()

ヘッダ	algorithm
書式	template<class iiter, class pred, class Dist> size_t count_if (iiter *first*, iiter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………条件を指定する述語。

●解説

このアルゴリズムは、特定の条件predを満たす要素数を返します。

●例

次の例はvectorに20個のランダムな数を保存し、5以上である要素数を調べる例です。

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool over5(int value)
{
    return value >4;
}

int main()
{
   vector<int> v;

   srand( (unsigned)time( NULL ) );

   // コンテナに値をプッシュする
   for(int i = 0 ; i < 20; ++i)
        v.push_back( rand() % 10 );

   for (int i=0; i< (int)v.size(); i++)
     cout << v[i] << ",";
   cout << endl;

   int n = count_if(v.begin(), v.end(), over5);

   cout << "5以上は" << n << "個" << endl;

   return 0;
}
```

● **実行例**

```
2,1,4,1,7,4,8,8,9,6,6,7,2,1,5,9,9,3,3,3,
5以上は10個
```

14	10進数に変換する dec
書式	ios& dec(ios_base& str);

● **解説**

このマニピュレータは、数値表示の基数を10進数に変更します。

● **例**

次の例は、値20を、16進数、10進数、8進数として表示します。

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int v = 20;

   cout << "hex=" << hex << v << endl;
   cout << "dec=" << dec << v << endl;
   cout << "oct=" << oct << v << endl;

   return 0;
}
```

15	先行する空白文字を取り出す eatwhite()
書式	void istream::eatwhite();

● **解説**

入力ポインタをスペースやタブの先へ進めて、ストリームから空白文字を取り除きます。これは以前のistream.hの関数です。新しいプログラムではistreamのマニピュレータwsを使ってください。

16 コンテナが空であるかどうか調べる empty()

書式	bool empty() const;

● 解説

シーケンスが空である場合にtrueを返します。

● **例**… stackの例参照。

17 最後の位置を指す反復子を返す end()

書式	① const_iterator Container::end() const; ② iterator Container::end();

● 解説

シーケンスの最後の直後の位置を指す反復子を返します。

● **例**… accumulate()、count()、count_if()の例参照。

18 改行文字を出力ストリームに出力する endl

書式	template class<E, T> basic_ostream<E, T>& endl(basic_ostream<E, T> os);

● 解説

endlは改行文字を出力ストリームに出力して、バッファをフラッシュするマニピュレータです。たとえば、ostream osに対しては、os.put(os. widen('¥n'))を呼び出し、さらにos.flush()を呼び出します。

● 例

次の例は、文字列"Hello!"および"Pochi."を出力したあとで改行する例です。

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   cout << "Hello!" << endl << "Pochi." << endl;

   return 0;
}
```

19 NULLバイトを出力ストリームに出力する ends

書式	template class<E, T> basic_ostream<E, T>& ends(basic_ostream<E, T> os);

解説

endsはNULL（¥0）を出力ストリームに出力するマニピュレータです。

20 2つの範囲を比較する equal()

ヘッダ	algorithm
書式	① template<class iiter1, class iiter2> bool equal(iiter1 *first*, iiter1 *last*, iiter2 *x*); ② template<class iiter1, class iiter2, class pred> bool equal (iiter1 *first*, iiter1 *last*, iiter2 *x*, pred *pr*); ③ bool istreambuf_iterator::equal(const istreambuf_iterator& rhs);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………比較する要素の反復子。 *pr* ………比較述語。

解説

書式①と②のアルゴリズムは、2つの範囲を比較して、等しければtrueを返します。

書式③の関数は、2つの反復子がNULLである（ストリームの最後である）か、両方がNULLでない（ストリームの最後でない）場合にtrueを返します。

21 コンテナの順序を変えずに要素を挿入できる最大範囲の位置を返す equal_range()

ヘッダ	algorithm
書式	① template<class iter, class T> pair<iter, iter> equal_range (iter *first*, iter *last*, const T& *val*); ② template<class iter, class T, class pred> pair<iter, iter> equal_range(iter *first*, iter *last*, const T& *val*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *val*………挿入する値。 *pr* ………各要素に適用する述語。

● **解説**

このアルゴリズムは、コレクションの順序を変えずに要素を挿入できる最大範囲の位置を表す反復子のペアを返します。

22 シーケンスから要素を消去する erase()

書式

① iterator Container::erase(iterator *it*);
② iterator Container::erase(iterator *first*, iterator *last*);
③ iterator basic_string::erase(iterator *first*, iterator *last*);
④ iterator basic_string::erase(iterator *it*);
⑤ basic_string& basic_string::erase(size_type p0 = 0, size_type n = npos);

引数

it ………削除する要素の反復子。
first ……最初の要素の反復子。
last ……最後の要素の反復子。

● **解説**

シーケンスから要素を消去します。

● **例**

次の例は、vectorから3、4、5番目の要素を削除する例です。

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

void print(const int p){
   cout << p << ", ";
}

int main(int argc, char *argv[])
{
   vector<int> v;

   // コンテナに値をプッシュする
   for(int i = 0 ; i < 10; ++i){
        v.push_back( i*2 );
   }
```

```
    // コンテナの要素に対して関数を適用する
    for_each( v.begin(), v.end(), print );
    cout << endl;

    v.erase(v.begin()+2, v.begin()+5);

    // コンテナの要素に対して関数を適用する
    for_each( v.begin(), v.end(), print );
    cout << endl;

    return 0;
}
```

● **実行例**

```
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
0, 2, 10, 12, 14, 16, 18,
```

23 コンテナを特定の値で初期化（再初期化）する fill()

ヘッダ	algorithm、ios、
書式	① template<class iter, class T> void fill(iter *first*, iter *last*, const T& *x*); ② E basic_ios::fill() const; ③ E basic_ios::fill(E ch);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………値。

● **解説**

書式①のアルゴリズムは、コンテナを特定の値で初期化あるいは再初期化します。

書式②の関数は、現在のフィルキャラクタを返します。③の関数は、フィルキャラクタとしてchを保存し、それ以前のフィルキャラクタの値を返します。

● 例

次の例は、vectorのサイズを7に設定してから、fill()を使って要素をすべて値0で初期化する例です。

```
#include <iostream>
#include <vector>

using namespace std;

int main(int argc, char* argv[])
{
   vector< int > v;

   v.resize(7);

   fill(v.begin(), v.end(), 0);

   for(int i=0; i<v.size(); i++)
      cout << v[i] << " ";

   cout << endl;

   return 0;
}
```

● 実行例

```
0 0 0 0 0 0 0
```

24 コンテナを特定の値で初期化（再初期化）する fill_n()

ヘッダ	algorithm
書式	template<class oiter, class Size, class T> void fill_n(oiter *first*, Size *n*, const T& *x*);
引数	*first* ……最初の要素の反復子。 *n* ………要素数。 *x* ………初期化する値。

解説

このアルゴリズムは、コンテナのn個の値を特定の値で初期化または再初期化します。

例

次の例は、fill_n()を使ってvectorの先頭（v.begin()）から3個の要素を値7で再初期化する例です。

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main(int argc, char* argv[])
{
   vector< int > v;
   int i;

   for(i=0; i<5; i++)
      v.push_back(i);

   for(i=0; i<v.size(); i++)
      cout << v[i] << " ";
   cout << endl;

   fill_n(v.begin(), 3, 7);

   for(i=0; i<v.size(); i++)
      cout << v[i] << " ";
   cout << endl;

   return 0;
}
```

実行例

```
0 1 2 3 4
7 7 7 3 4
```

25 コンテナの要素を検索する find()

ヘッダー	algorithm
書式	① template<class iiter, class T> iiter find(iiter *first*, iiter *last*, const T& *val*); ② iterator find(const Key& *key*); ③ const_iterator find(const Key& *key*) const;
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *val*………検索する値。 *key* ……検索するキー。

●解説

コンテナの要素を検索するアルゴリズムです（書式①）。また、set、multiset、map、multimapなどのSTLコンテナのメンバ関数としても定義されています（書式②）。

●例

次の例はstringのリスト（list）であるnamelistに保存した名前を検索する例です。

```
#include <iostream>
#include <string>
#include <list>
#include <algorithm>

#define BUFF_LEN 1024

using namespace std;

list<string> namelist;

// .endが入力されるまで、繰り返し名前を入力する
int GetNames()
{
   int n = 0;
   char buff[BUFF_LEN];
   while(1)
   {
      cout << "名前は? >";
      cin.getline(buff, BUFF_LEN);
      if (string(buff).compare(".end") == 0)
         break;
      namelist.push_back(string(buff));
```

```
      n++;
   }
   return n;
}

bool FindName()
{
   char buff[BUFF_LEN];
   cout << "探す名前は? >";
   cin.getline(buff, BUFF_LEN);
   list<string>::iterator s;
   s = find(namelist.begin(), namelist.end(), string(buff));
   if (s != namelist.end())
   {
      cout << *s << "は登録済みです。" << endl;
      return true;
   } else
      cout << buff << "は登録されていません。" << endl;
   return false;
}

int main(int argc, char *argv[])
{
   GetNames();
   FindName();
   return 0;
}
```

26 コンテナの中のサブシーケンスの最後の要素を検索する find_end()

ヘッダ	algorithm
書式	① template<class iter1, class iter2> iter1 find_end(iter1 *first1*, iter1 *last1*, iter2 *first2*, iter2 *last2*); ② template<class iter1, class iter2, class pred> iter1 find_end (iter1 *first1*, iter1 *last1*, iter2 *first2*, iter2 *last2*, pred *pr*);
引数	*first1* …探すコンテナの最初の要素の反復子。 *last1* ……探すコンテナの最後の要素の反復子。 *first2* …探すサブシーケンスの最初の要素の反復子。 *last2* ……探すサブシーケンスの最後の要素の反復子。 *pr* ………比較述語。

● **解説**… このアルゴリズムは、コンテナの中のサブシーケンスの最後の要素を検索します。

● **参照**→find()

27 コンテナの中のサブシーケンスの最初の要素を検索する find_first_of()

ヘッダ	algorithm
書式	① template<class iter1, class iter2> iter1 find_first_of (iter1 *first1*, iter1 *last1*, iter2 *first2*, iter2 *last2*); ② template<class iter1, class iter2, class pred> iter1 find_first_of (iter1 *first1*, iter1 *last1*, iter2 *first2*, iter2 *last2*, pred *pr*); ③ size_type basic_string::find_first_of(E c, size_type *pos* = 0) const; ④ size_type basic_string::find_first_of(const E *s, size_type *pos* = 0) const; ⑤ size_type basic_string::find_first_of(const E *s, size_type *pos*, size_type *n*) const; ⑥ size_type basic_string::find_first_of(const basic_string& str, size_type *pos* = 0) const;
引数	*first1* …探すコンテナの最初の要素の反復子。 *last1* ……探すコンテナの最後の要素の反復子。 *first2* …探すサブシーケンスの最初の要素の反復子。 *last2* ……探すサブシーケンスの最後の要素の反復子。 *pr* ………比較述語。 *pos* ……探す最初の位置。 *n* ………探す要素数。

●解説

書式①と②のアルゴリズムは、コンテナの中のサブシーケンスの最初の要素を検索します。

書式③〜⑥の関数は、制御シーケンスの位置*pos*よりあとにあり、検索する要素のいずれかに一致する最初の要素を探します。検索に失敗したときはnposを返します。

28 特定の条件を満足させる最初の要素を検索する find_if()

ヘッダ	algorithm
書式	template<class iiter, class pred> iiter find_if(iiter *first*, iiter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………検索条件を指定する述語。

●解説

このアルゴリズムは、特定の条件*pr*を満足させる最初の要素を検索します。

●例

次の例は、vectorに保存した20個の中から最初の偶数を探すプログラムの例です。

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool isEven(int n)
{
   if (n % 2)
      return false;
   return true;
}

int main()
{
   vector<int> v;

   srand( (unsigned)time( NULL ) );
```

```
    // コンテナに値をプッシュする
    for(int i = 0 ; i < 20; ++i)
        v.push_back( rand() % 10 );

    for (int i=0; i< (int)v.size(); i++)
      cout << v[i] << ",";
    cout << endl;

    vector <int>::iterator it;
    it = find_if(v.begin(), v.end(), isEven);

    cout << "最初の偶数は" << (*it) << endl;

    return 0;
}
```

● **実行例**

```
3,1,8,7,5,8,9,3,3,6,6,3,3,7,2,0,2,4,4,5,
最初の偶数は8
```

29 出力ストリームをフラッシュする flush

書式	① ostream& ostream::flush(); ② basic_ostream& basic_ostream::flush();

● **解説**

このマニピュレータは、出力ストリームのクラスの関数sync()を呼び出して、出力ストリームをフラッシュします。

30 コレクションの要素に関数を適用する for_each()

ヘッダ	<algorithm>
書式	template<class iiter, class Fun> Fun for_each(iiter *first*, iiter *last* Fun *f*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *f* ………各要素に適用する関数。

解説

このアルゴリズムは、コレクションの中の反復子*first*から*last*の範囲のすべての要素に、引数で指定した関数を適用します。

例

次の例は、for_eachを使ってvectorの各要素に対してprint()を適用し、すべての要素を出力する例です。

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

void print(const int p){
   cout << p << endl;
}

int main(int argc, char *argv[])
{
   vector<int> v;

   // コンテナに値をプッシュする
   for(int i = 0 ; i < 4; ++i){
        v.push_back( i*2 );
   }

   // コンテナの要素に対して関数を適用する
   for_each( v.begin(), v.end(), print );

   return 0;
}
```

実行例

```
0
2
4
6
```

31 シーケンスの最初の要素を指す参照を返す front()

書式	① reference QSContainer::front(); ② const_reference QSContainer::front() const;

● 解説

空ではないシーケンスの最初の要素を指す参照を返します。

32 ファイルストリームを作成する fstream()

ヘッダ	fstream
書式	① fstream::fstream(); ② fstream::fstream(const char* *fName*, int *nMode*, int *nProtect* = filebuf::openprot); ③ fstream::fstream(filedesc *fd*);
引数	*fName* ………ファイル名。 *nMode* ………ファイルモード。 *nProtect* ……プロテクトモード。 *fd* ……………ファイルデスクリプタ。

● 解説

このコンストラクタは、fstreamオブジェクトを構築します。引数を指定しない書式①の場合、ファイルを開かずに、fstreamオブジェクトを構築します。ファイル名*fName*またはファイルディスクリプタ*fd*を指定すると、指定されたファイルを開きます。

ファイルを開くときのモード*nMode*には、以下の列挙子をOR（|）演算子を使って組み合わせて指定することができます。

▼表 ファイルのモード

フラグ	意味
ios::app	ファイルの終わりに移動し、以降の操作で書き込まれるバイトは必ずファイルの終わりに追加される。
ios::ate	ファイルの終わりに移動し、最初に新しいバイトが書き込まれるときはファイルの終わりに追加され、その後に続くバイト列は現在位置に書き込まれる。
ios::in	ファイルを入力用に開く（既存のファイルは削除されない）。
ios::out	ファイルを出力用に開く。
ios::trunc	ファイルが存在している場合、内容が破棄される。
ios::nocreate	ファイルが存在しないと、関数はエラーを発生する。
ios::noreplace	ファイルが存在すると、関数はエラーを発生する。
ios::binary	ファイルをバイナリモードで開く（デフォルトはテキストモード）。

*nProtect*にはファイルの保護方法を指定します。

▼表 nProtectの値

nProtectの値	意味
filebuf::sh_compat	互換共有モード（Windows環境でMS-DOSの場合）
filebuf::sh_none	排他モード（共有しない）
filebuf::sh_read	読み込み共有許可モード
filebuf::sh_write	書き込み共有許可モード

●例

次の例は、ファイルを開いて最後の16バイトを出力するプログラムの例です。

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

#define FILENAME "./test.dat"

int main(int argc, char *argv[])
{
   char ch;
   fstream fs(FILENAME, ios::in || ios::binary );
   if( !fs )
   {
      cout << "ファイル" << FILENAME << "を開けません" << endl;
      return 1;
   }

   // 最後から16バイト前の位置に移動する
   fs.seekg(-16, ios::end);

   // EOFかエラーが発生するまで読み込んで表示する
   while ( fs.good() ) {
      ch = 0;
      fs.get( ch );
      if (ch)
         cout << ch;
      else
         break;
   }
   return 0;
}
```

33 直前の入力関数で抽出された文字数を返す gcount()

ヘッダ	istream
書式	① streamsize gcount() const; ② int istream::gcount() const;

● 解説

最後に呼び出された書式なし入力関数によって抽出された文字数を返します。

書式②は以前のIOstreamライブラリ（iostream.h）の関数です。

34 ジェネレータで生成した値でコンテナを初期化（再初期化）する generate()

ヘッダ	algorithm
書式	template<class iter, class Gen> void generate(iter *first*, iter *last*, Gen *g*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *g* ………ジェネレータ関数。

● 解説

このアルゴリズムは、コンテナの反復子*first*と*last*で指定した範囲の値を、ジェネレータ*g*で生成した値で初期化あるいは再初期化します。

● 例

次の例は、アルゴリズムgenerate()を使ってコンテナを初期化する例です。この例ではジェネレータ関数の名前はgenerator()ですが、ほかの名前でもかまいません。

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int generator()
{
   static int v = 1;
   return v++;
}
```

```
int main(int argc, char* argv[])
{
   vector< int > v;
   v.resize(7);
   generate(v.begin(), v.end(), generator);

   for(int i=0; i<v.size(); i++)
      cout << v[i] << " ";
   cout << endl;

   return 0;
}
```

35 ジェネレータで生成した値でコンテナを初期化(再初期化)する generate_n()

ヘッダ	algorithm
書式	template<class oiter, class pred, class Gen> void generate_n (oiter *first*, Dist *n*, Gen *g*);
引数	*first* ……最初の要素の反復子。 *n* ………要素数。 *g* ………ジェネレータ関数。

● 解説

このアルゴリズムは、ジェネレータ*g*で生成した値でコンテナの要素*n*個を初期化あるいは再初期化します。

● 参照→generate()

36 データを取得する get()

ヘッダ istream

書式
```
① int_type basic_istream::get();
② istream_type& istream::get(char_type& c);
③ istream_type& istream::get(char_type* s, streamsize n,
   char_type delim);
④ istream& istream::get( streambuf& rsb, char delim = '¥n' );
⑤ string_type messages::get(catalog cat, int set, int msg,
   const string_type& dflt) const;
⑥ T *auto_ptr::get() const throw();
⑦ iter_type money_get::get(iter_type first, iter_type last, bool intl,
   ios_base& x, ios_base::iostate& st, long double& val) const;
⑧ iter_type money_get::get(iter_type first, iter_type last, bool intl,
   ios_base& x, ios_base::iostate& st, string_type& val) const;
```

解説

get()はさまざまなクラスでデータを取得します。たとえば、istreamでは、デリミタ(区切り文字)があるまでストリームから文字を取り出します。デフォルトの区切り文字は¥nです。

istreamの引数なしの書式は、文字があればそれを取得します。

例… seekg()の参照。

37 ストリームから行単位で文字列を入力する getline()

ヘッダ string

書式
```
① basic_istream::getline()
② basic_istream& getline(E *s, streamsize n);
③ basic_istream& getline(E *s, streamsize n, [E delim]);
④ istream& getline( char* pch, int n, char delim = '¥n' );
⑤ istream& getline( unsigned char* puch, int n, char delim = '¥n' );
⑥ istream& getline( signed char* psch, int n, char delim = '¥n' );
⑦ istream_type& getline(char_type* s, streamsize n,
   char_type delim);
```

● 解説

文字を取得します。読み込む文字数が指定されている場合は、最大でその文字数までがバッファに保存されます。そうでなければ、EOF（End of File）に達するまで、あるいはデリミタ（区切り文字）delimまで読み込まれます。書式①はgetline(s, n, widen('¥n'))を返します。

● 例

次の例は、“.end”が入力されるまで、リストに繰り返し名前を保存する例です。この例では、関数GetNames()の中でgetline()で得た入力文字列を繰り返し入力して、入力文字列が“.end”であった場合、ループを抜けます。その後、DispNames()では反復子を使ってリストの内容を出力します。

```
#include <iostream>
#include <string>
#include <list>

#define BUFF_LEN 1024

using namespace std;

list<string> namelist;

// .endが入力されるまで、繰り返し名前を入力する
int GetNames()
{
    int n = 0;
    char buff[BUFF_LEN];
    while(1)
    {
        cout << "名前は? >";
        cin.getline(buff, BUFF_LEN);
        if (string(buff).compare(".end") == 0)
            break;
        namelist.push_back(string(buff));
        n++;
    }
    return n;
}

void DispNames()
{
```

```
    list<string>::iterator it = namelist.begin();
    for (; it != namelist.end(); it++)
    {
        cout << *it << endl;
    }
}

int main(int argc, char *argv[])
{
    GetNames();
    DispNames();
    return 0;
}
```

38 入出力ストリームのエラー状態を返す good()

ヘッダ	ios
書式	① bool basic_ios::good() const; ② int ios::good() const;

● **解説**

エラーのフラグがセットされていないときに0以外の値を返します。

● **例**… seekg()の例参照。

39 16進数に変換する hex

書式	ios& hex(ios_base& str);

● **解説**

このマニピュレータは、数値表示の基数を16進数に変更します。

● **例**… decの例参照。

40	ストリームから文字を取り出して廃棄する **ignore()**
ヘッダ	istream
書式	① istream& istream::ignore(int *n* = 1, int *delim* = EOF); ② basic_istream& basic_istream::ignore(streamsize *n* = 1, int_type *delim* = T::eof());
引数	*n* ………破棄する文字数。 *delim* …デリミタ文字。

● 解説

文字を取得して破棄します。指定されている最大文字数かEOFに達するまで、あるいはデリミタ（区切り文字）*delim*に達するまで文字がストリームから取り出されます。取り出された文字は破棄されるので、プログラムはそれらの文字を無視することになります。

41	一方の範囲のすべての要素が他の範囲に含まれているか調べる **includes()**
ヘッダ	algorithm
書式	① template<class iiter1, class iiter2> bool includes(iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*); ② template<class iiter1, class iiter2, class pred> bool includes (iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, pred *pr*);
引数	*first1* …調べるコンテナの最初の要素の反復子。 *last1* ……調べるコンテナの最後の要素の反復子。 *first2* …調べるサブシーケンスの最初の要素の反復子。 *last2* ……調べるサブシーケンスの最後の要素の反復子。 *pr* ………比較述語。

● 解説

このアルゴリズムは、2つのソートされたコンテナを比較して、一方の範囲のすべての要素が、もう1つの範囲に含まれている場合にtrueを返します。

42 コンテナをマージする inplace_merge()

ヘッダ	algorithm
書式	① template<class biter> void inplace_merge(biter *first*, biter *middle*, biter *last*); ② template<class biter, class pred> void inplace_merge (biter *first*, biter *middle*, biter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *middle*…中央の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較述語。

● 解説

このアルゴリズムは、*first*～*middle*と*middle*～*last*の範囲をマージして1つにします。

● 参照→merge()

43 シーケンスに要素を挿入する insert()

書式	① iterator Container::insert(iter *it*, const T& *x* = T()); ② void Container::insert(iter *it*, size_type *n*, const T& *x*); ③ void Container::insert(iter *it*, const_iter *first*, const_iter *last*);
引数	*it* ………挿入する位置を示す反復子。 *x* ………挿入する値。 *n* ………挿入する要素数。 *first* ……挿入する最初の反復子。 *last* ……挿入する最後の反復子。

● 解説

シーケンスに要素を挿入します。

●例

次の例は、vector::insert()を使ってコンテナの先頭に文字を挿入する方法を2種類示す例です。

```
#include <iostream>
#include <vector>

using namespace std;

int main(int argc, char* argv[])
{
   vector< char > v;
   v.push_back('a');
   v.push_back('b');
   v.push_back('c');

   // 1個だけ挿入
   v.insert(v.begin(), 'x');

   // 2個挿入
   v.insert(v.begin(), 2, 'y');

   for(int i=0; i<v.size(); i++)
     cout << v[i] << " ";
   cout << endl;

   return 0;
}
```

●実行例

```
y y x a b c
```

44 ストリーム入力の準備をする ipfx()

書式	① int istream::ipfx(int need = 0); ② bool basic_istream::ipfx(bool noskip = false);

● 解説

ストリーム入力のための準備を行います。これらの関数はストリームからデータを取り出す前に入力関数が呼び出します。プログラマが直接呼び出すことはめったにありません。

45 ストリームからの入力終了の処理をする isfx()

書式	① void istream::isfx(); ② void basic_istream::isfx();

● 解説

ストリームから入力関数が、入力の操作を終了するたびに呼び出します。

46 2つの反復子が指し示す値を交換する iter_swap()

ヘッダ	algorithm
書式	template<class iter1, class iter2> void iter_swap(iter1 *x*, iter2 *y*);
引数	*x*、*y* ……交換する要素を指す反復子。

● 解説

このアルゴリズムは、2つの反復子が指し示す値を交換します。

● 例

次の例はvectorの3番めと6番めの要素を入れ替える例です。

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
   vector<int> v;
```

```
    // コンテナに値をプッシュする
    for(int i = 0 ; i < 10; ++i)
        v.push_back( i );

    // コンテナの内容を出力する
    cout << "v=";
    copy(v.begin(), v.end(), ostream_iterator<int,char>(cout,","));
    cout << endl;

    // 3番めと6番めを入れ替える
    iter_swap(v.begin()+2, v.begin()+5);

    cout << "v=";
    copy(v.begin(), v.end(), ostream_iterator<int,char>(cout,","));
    cout << endl;

    return 0;
}
```

● 実行例

```
v=0,1,2,3,4,5,6,7,8,9,
v=0,1,5,3,4,2,6,7,8,9,
```

47 要素の順序を決定する比較関数オブジェクトを返す key_comp()

書式	key_compare AssoCont::key_comp() const;

● 解説

連想コンテナの要素の順序を決定する比較関数オブジェクトを返します。

48 2つの範囲を辞書順で比較する lexicographical_compare()

ヘッダ	algorithm
書式	① template<class iiter1, class iiter2> bool lexicographical_compare(iiter1 first1, iiter1 last1, iiter2 first2, iiter2 last2); ② template<class iiter1, class iiter2, class pred> bool lexicographical_compare(iiter1 first1, iiter1 last1, iiter2 first2, iiter2 last2, pred pr);

● 解説

このアルゴリズムは、2つの範囲を辞書順で比較します。

● 参照→compare()

49 コンテナの値が入る下限を示す反復子を返す lower_bound()

書式	① template<class iter, class T> iter lower_bound(iter first, iter last, const T& val); ② template<class iter, class T, class pred> iter lower_bound(iter first, iter last, const T& val, pred pr); ③ iterator AssoCont::lower_bound(const Key& key); ④ const_iterator AssoCont::lower_bound(const Key& key) const;

● 解説

書式①と②のアルゴリズムは、ソート済みのコンテナで、値が入る下限を示す反復子を返します。書式③以降の関数はkey_comp()(x, key)がfalseである最初の要素を指す反復子を返します。

50 ヒープを作成する make_heap()

ヘッダ	algorithm
書式	① template<class riter> void make_heap(riter first, riter last); ② template<class riter, class pred> void make_heap(riter first, riter last, pred pr);

● 解説

このアルゴリズムは、ヒープを作成します。

例

次の例はヒープを作成して操作する例です。

```
#include <algorithm>
#include <vector>
#include <iostream>

using namespace std;

int main(void)
{
   vector<int> v1;
   for (int i=0; i<5; i++)
      v1.push_back(i+1);

   make_heap(v1.begin(),v1.end());

   // コンテナの内容を出力する
   ostream_iterator<int,char> out(cout," ");
   copy(v1.begin(), v1.end(), out);
   cout << endl;

   pop_heap(v1.begin(),v1.end());
   // コンテナの内容を出力する
   cout << "pop_heap後" << endl;
   copy(v1.begin(),v1.end(),out);
   cout << endl;

   push_heap(v1.begin(),v1.end());
   // コンテナの内容を出力する
   cout << "push_heap後" << endl;
   copy(v1.begin(),v1.end(),out);
   cout << endl;

   sort_heap(v1.begin(),v1.end());
   // コンテナの内容を出力する
   cout << "sort_heap後" << endl;
   copy(v1.begin(),v1.end(),out);
   cout << endl;

   return 0;
}
```

● 実行例

```
5 4 3 1 2
pop_heap後
4 2 3 1 5
push_heap後
5 4 3 1 2
sort_heap後
1 2 3 4 5
```

51 最大値を返す max()

書式

① template<class T> const T& max(const T& x, const T& y);

② template<class T, class pred> const T& max(const T& x, const T& y, pred pr);

③ template<class T> T max(const valarray<T>& x);

④ static T numeric_limits::max() throw();

⑤ T valarray::max() const;

● 解説

このアルゴリズムは、2つの値の最大値を返します。

● 例

次の例は、それぞれ2個の整数と文字列の最大値を出力する例です。

```
#include <iostream>
#include <string>

using namespace std;

int main(void)
{
   int v1=12, v2=30;

   cout << max(v1, v2) << endl;

   string s1="ABC", s2="xyz";

   string s3 = max(s1, s2);
```

```
    cout << s3 << endl;

    return 0;
}
```

実行例

```
30
xyz
```

52 コンテナの中の最大の要素を示す反復子を返す max_element()

ヘッダ	algorithm
書式	① template<class iter> iter max_element(iter *first*, iter *last*); ② template<class iter, class pred> iter max_element(iter *first*, iter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較のための述語。

解説

このアルゴリズムは、コンテナの中の最大の要素を示す反復子を返します。

例

次の例はmax_element()とmin_element()を使って要素の値が最大のものと最小のものを探して、その反復子を取得します。取得した反復子に対して演算子*を適用することでその反復子が示す値を出力します。

アルファベットの文字コードは大文字のほうが小さいので、最小値が大文字のFであることに注意してください。

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main(int argc, char* argv[])
{
   vector< char > v;
   vector< char >::const_iterator itermax, itermin;

   v.push_back('a');
   v.push_back('c');
   v.push_back('z');
   v.push_back('F');

   itermax = max_element(v.begin(), v.end());
   cout << "最大要素=" << (*itermax) << endl;

   itermin = min_element(v.begin(), v.end());
   cout << "最小要素=" << (*itermin) << endl;

   return 0;
}
```

●実行例

```
最大要素=z
最小要素=F
```

53 要素数の上限を返す max_size()

書式	size_type Container::max_size() const;

解説

利用できる要素の最大の数を返します。

例

次の例はvectorの利用できる要素の最大の数を返すプログラムの例です。

```
#include <iostream>
#include <vector>

using namespace std;

int main(void)
{
   vector <int> v;

   for (int i=0; i<5; i++)
      v.push_back(1);

   cout << v.max_size() << endl;

   return 0;
}
```

実行例

```
1073741823
```

54 2つのソートされたコンテナをマージする merge()

ヘッダ	algorithm
書式	① template<class iiter1, class iiter2, class oiter> oiter merge (iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*); ② template<class iiter1, class iiter2, class oiter, class pred> oiter merge(iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*, pred *pr*); ③ void list::merge(list& *x*); ④ void list::merge(list& *x*, greater<T> *pr*);
引数	*first1* …マージする1個目のコンテナの最初の要素の反復子。 *last1* ……マージする1個目のコンテナの最後の要素の反復子。 *first2* …マージする第二のコンテナの最初の要素の反復子。 *last2* ……マージする第二のコンテナの最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の反復子。 *pr* ………各要素に適用する述語。

●解説

書式①と②のアルゴリズムは、2つのソートされたコンテナをマージして、別のコンテナに入れます。

書式③と④の関数は、引数のリストをマージします。

●例

次の例は、2個のリストを作成して、第3のリストにマージする例です。

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <list>
#include <algorithm>

using namespace std;

void print(const int p){
   cout << p << ", ";
}

int main(int argc, char *argv[])
{
   list<int> lst1, lst2, lst3(10);
```

```
    srand( (unsigned)time( NULL ) );

    // コンテナに値をプッシュする
    for(int i = 0 ; i < 5; ++i){
        lst1.push_back( rand() % 100 );
    }

    // コンテナに値をプッシュする
    for(int i = 0 ; i < 3; ++i){
        lst2.push_back( rand() % 100 );
    }

    // コンテナの要素を出力する
    cout << "lst1:";
    for_each( lst1.begin(), lst1.end(), print );
    cout << endl;
    cout << "lst2:";
    for_each( lst2.begin(), lst2.end(), print );
    cout << endl;

    lst1.sort();
    lst2.sort();

    // コンテナの要素を出力する
    cout << "lst1(sorted):";
    for_each( lst1.begin(), lst1.end(), print );
    cout << endl;
    // コンテナの要素を出力する
    cout << "lst2(sorted):";
    for_each( lst2.begin(), lst2.end(), print );
    cout << endl;

    merge(lst1.begin(), lst1.end(), lst2.begin(), lst2.end(), lst3.begin());

    // コンテナの要素を出力する
    cout << "lst3:";
    for_each( lst3.begin(), lst3.end(), print );
    cout << endl;

    return 0;
}
```

● 実行例

```
lst1:43, 74, 48, 60, 74,
lst2:32, 68, 89,
lst1(sorted):43, 48, 60, 74, 74,
lst2(sorted):32, 68, 89,
lst3:32, 43, 48, 60, 68, 74, 74, 89, 0, 0,
```

55 最小値を返す min()

書式	max()参照。

● 解説

このアルゴリズムは、2つの値の最小値を返します。max()を参照してください。

56 コンテナの中の最小の要素を示す反復子を返す min_element()

ヘッダ	algorithm
書式	① template<class iter> iter min_element(iter first, iter last); ② template<class iter, class pred> iter min_element(iter first, iter last, pred pr);

● 解説

このアルゴリズムは、コンテナの中の最小の要素を示す反復子を返します。

● **例**… max_element()の例参照。

57 2つのコンテナの一致しない最初の要素を返す mismatch()

ヘッダ	algorithm
書式	① template<class iiter1, class iiter2> pair<iiter1, iiter2> mismatch (iiter1 *first*, iiter1 *last*, iiter2 *x*); ② template<class iiter1, class iiter2, class pred> pair<iiter1, iiter2> mismatch(iiter1 *first*, iiter1 *last*, iiter2 *x*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………比較するコンテナの反復子。 *pr* ………比較に使う述語。

●解説

このアルゴリズムは、2つのコンテナの要素を比較して一致しない最初の要素の範囲を示す反復子のペアを返します。

●例

次の例は、2個のvectorの要素を比較して、一致しない最初の要素を返します。

```
#include <algorithm>
#include <vector>
#include <iostream>

using namespace std;

int main(void)
{
   typedef vector<int>::iterator iterator;

   int d1[] = {1, 2, 3, 4, 5};
   vector<int> v1(d1, d1 + 5);

   int d2[] = {1, 3, 4, 4, 5};
   vector<int> v2(d2,d2 + 5);

   // コンテナの内容を出力する
   ostream_iterator<int,char> out(cout, ",");
   cout << "v1:" << endl;
   copy(v1.begin(), v1.end(), out);
   cout << endl;
   cout << "v2:" << endl;
   copy(v2.begin(), v2.end(), out);
   cout << endl;

   // 一致しない要素を探す
   pair<iterator, iterator> p;
   p = mismatch(v1.begin(), v1.end(), v2.begin());
   // 結果を出力する
   cout << (*p.first) << ", " << (*p.second) << endl;

   return 0;
}
```

● 実行例

```
v1:
1,2,3,4,5,
v2:
1,3,4,4,5,
2, 3
```

58 順序付け関数に従って連続した順列を生成する next_permutation()

ヘッダ	algorithm
書式	① template<class biter> bool next_permutation(biter *first*, biter *last*); ② template<class biter, class pred> bool next_permutation (biter *first*, biter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較に使う述語。

● 解説

このアルゴリズムは、順序付け関数に従って連続した順列を生成します。

● 例

次の例は、vectorの要素を大きい順（降順）に並べ替える例です。

```
#include <algorithm>
#include <vector>
#include <iostream>
#include <functional>

using namespace std;

int main(void)
{
   typedef vector<int>::iterator iterator;

   int d[] = {11, 8, 6, 4, 7};
   vector<int> v(d, d + 5);
```

```
    // コンテナの内容を出力する
    ostream_iterator<int,char> out(cout, ",");
    cout << "before:" << endl;
    copy(v.begin(), v.end(), out);
    cout << endl;

    prev_permutation(v.begin(), v.end(), less<int>());

    // 結果を出力する
    cout << "after:" << endl;
    copy(v.begin(), v.end(), out);
    cout << endl;

    return 0;
}
```

●実行例

```
before:
11,8,6,4,7,
after:
11,8,4,7,6,
```

59 比較演算子を使ってコレクションを再配置する nth_element()

書式	① template<class riter> void nth_element(riter *first*, riter *nth*, riter *last*); ② template<class riter, class pred> void nth_element(riter *first*, riter *nth*, riter *last*, pred pr);
引数	*first* ……最初の要素の反復子。 *nth* ……n番目の要素の反復子。 *last* ……最後の要素の反復子。

●解説

このアルゴリズムは、ソートされた順序でn番目の要素より低い要素がすべてその前に来て、ソートされた順序でn番目の要素より高い要素がすべてそのあとに来るようにコレクションの要素を再配置します。

例

次の例は、vectorの要素を並べ替える例です。

```
#include <algorithm>
#include <vector>
#include <iostream>
#include <functional>

using namespace std;

int main(void)
{
   typedef vector<int>::iterator iterator;

   int d[] = {11, 8, 6, 4, 7, 3, 10};
   vector<int> v(d, d + 7);

   // コンテナの内容を出力する
   ostream_iterator<int,char> out(cout, ",");
   cout << "before:";
   copy(v.begin(), v.end(), out);
   cout << endl;

   nth_element(v.begin(), v.begin()+3, v.end());

   // 結果を出力する
   cout << "after:";
   copy(v.begin(), v.end(), out);
   cout << endl;

   return 0;
}
```

実行例

```
before:11,8,6,4,7,3,10,
after:3,4,6,7,8,10,11,
```

60	8進数に変換する oct
書式	ios& oct(ios_base& str);

解説

このマニピュレータは、数値表示の基数を8進数に変更します。

例… decの例参照。

61	オブジェクトを開く open()
ヘッダ	fstream
書式	① void fstream::open(const char* *fName*, int *nMode*, int *nProtect* = filebuf::openprot); ② basic_filebuf *basic_filebuf::open(const char *s, ios_base::openmode mode); ③ void basic_fstream::open(const char *s, ios_base::openmode mode = ios_base::in | ios_base::out); ④ void basic_ifstream::open(const char *s, ios_base::openmode mode = ios_base::in); ⑤ void basic_ofstream::open(const char *s, ios_base::openmode mode = ios_base::out | ios_base::trunc); ⑥ catalog messages::open(const string& name, const locale& loc) const;
引数	*fName*………ファイル名。 *nMode* ……ファイルモード。 *nProtect* ……プロテクトモード。

解説

fstreamクラスのopenはディスク上のファイル*fName*を開き、ストリームのオブジェクトに割り当てます。*nMode*と*nProtect*はfstream()を参照してください。

書式⑥はメッセージカタログを開きます。

●例

次の例は、既存のファイルを開いて読み込み、標準出力に出力するプログラムの例です。

```
#include <algorithm>
#include <fstream >
#include <iostream>

using namespace std;

int main()
{
   fstream fs;

   fs.open("test.txt", ios_base::in);

   char c;

   while ( !fs.eof() )
   {
      fs.read(&c, 1);
      cout << c;
   }

   return 0;
}
```

● 参照→fstream

62 ストリームへデータを出力する operator <<

書式	ostream& ostream::operator <<(type v);

●解説

さまざまな型のデータを出力ストリームに出力します。

●例

次の例は、文字列と整数値を標準入力から入力して、標準出力に出力する例です。

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char *argv[])
{
   string s;
   int v;
   cout << "文字列>";
   cin >> s;
   cout << "整数>";
   cin >> v;
   cout << "文字列=" << s << " 整数=" << v <<endl;

   return 0;
}
```

63 ストリームからデータを読み込む operator >>

書式	① istream& istream::operator >>(type* pv); ② istream& istream::operator >>(type& rv);

●解説

さまざまな型のデータを、入力ストリームからポインタpvまたは参照rvに読み込みます。

●例

operator <<の例、wsの例参照。

64 指定した範囲の値をソートする partial_sort()

ヘッダ	algorithm
書式	① template<class riter> void partial_sort(riter *first*, riter *middle*, riter *last*); ② template<class riter, class pred> void partial_sort(riter *first*, riter *middle*, riter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *middle*…中間の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較に使う述語。

● 解説

このアルゴリズムは、*first*から*middle*の範囲の値をソートします。

● 例

次の例は、7個の要素があるvectorの最初から5個目までをソートするプログラムの例です。

```
#include <algorithm>
#include <vector>
#include <iostream>
#include <functional>

using namespace std;

int main(void)
{
   typedef vector<int>::iterator iterator;

   int d[] = {11, 8, 6, 4, 7, 3, 10};
   vector<int> v(d, d + 7);

   // コンテナの内容を出力する
   ostream_iterator<int,char> out(cout, ",");
   cout << "before:";
   copy(v.begin(), v.end(), out);
   cout << endl;

   partial_sort(v.begin(), v.begin() + 4, v.end());

   // 結果を出力する
   cout << "after:";
   copy(v.begin(), v.end(), out);
   cout << endl;

   return 0;
}
```

● 実行例

```
before:11,8,6,4,7,3,10,
after:3,4,6,7,11,8,10,
```

● 参照→partial_sort_copy()

65 指定した範囲の値をソートして別のコンテナに保存する partial_sort_copy()

ヘッダ	algorithm
書式	① template<class iiter, class riter> riter partial_sort_copy (iiter *first1*, iiter *last1*, riter *first2*, riter *last2*); ② template<class iiter, class riter, class pred> riter partial_sort_copy(iiter *first1*, iiter *last1*, riter *first2*, riter *last2*, pred *pr*);
引数	*first1* …最初の要素の反復子。 *last1* ……最後の要素の反復子。 *first2* …コピーするコンテナの最初の要素の反復子。 *last2* ……コピーするコンテナの最後の要素の反復子。 *pr* ………比較に使う述語。

● 解説

このアルゴリズムは、指定した範囲の値をソートして、結果を別のコンテナにコピーします。

● 例

次の例は、v1という名前のvectorの要素をソートしてv2に保存するプログラムの例です。

```
#include <algorithm>
#include <vector>
#include <iostream>
#include <functional>

using namespace std;

int main(void)
{
   typedef vector<int>::iterator iterator;
```

```
    int d[] = {11, 8, 6, 4, 7, 3, 10};
    vector<int> v1(d, d + 7);
    vector<int> v2(d, d + 7);

    // コンテナの内容を出力する
    ostream_iterator<int,char> out(cout, ",");
    cout << "v1:";
    copy(v1.begin(), v1.end(), out);
    cout << endl;

    partial_sort_copy(v1.begin(), v1.end(), v2.begin(), v2.end());

    // 結果を出力する
    cout << "v2:";
    copy(v2.begin(), v2.end(), out);
    cout << endl;

    return 0;
}
```

● **実行例**

```
v1:11,8,6,4,7,3,10,
v2:3,4,6,7,8,10,11,
```

● **参照**→partial_sort()

66 条件を満足させる要素を、条件を満足しない要素より前に配置する partition()

ヘッダ	algorithm
書式	template<class biter, class pred> biter partition(biter *first*, biter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較に使う述語。

● **解説**

このアルゴリズムは、条件を満足させる要素を、条件を満足しない要素より前に配置します。

●例

次の例は、vectorの要素のうち、偶数である要素を前に集めるプログラムの例です。

```
#include <algorithm>
#include <vector>
#include <iostream>

using namespace std;

bool is_even(int v)
{
   return (v % 2) == 0;
}

int main(void)
{
   typedef vector<int>::iterator iterator;

   int d[] = {11, 8, 6, 4, 7, 3, 10};
   vector<int> v(d, d + 7);

   // コンテナの内容を出力する
   ostream_iterator<int,char> out(cout, ",");
   cout << "before:";
   copy(v.begin(), v.end(), out);
   cout << endl;

   //partition(v.begin(), v.end(), is_even<int>());
   partition(v.begin(), v.end(), is_even);

   // 結果を出力する
   cout << "after:";
   copy(v.begin(), v.end(), out);
   cout << endl;

   return 0;
}
```

●実行例

```
before:11,8,6,4,7,3,10,
after:10,8,6,4,7,3,11,
```

67 ストリームから文字を取り出さずに文字を返す peek()

ヘッダ	istream
書式	① int_type basic_istream::peek(); ② int_type istream::peek();

●解説

ストリームから文字を取り出さずに次の文字を返します。文字を取り出せなかったストリームがファイルの終わりに達するとEOFあるいはtraits::eof()を返します。

●例

次の例は、ファイルの先頭の文字を取り出して出力します。

```
#include <fstream >
#include <iostream>

using namespace std;

int main()
{
   fstream fs;
   fs.open("test.txt");

   char c = fs.peek();
   cout << c;

   return 0;
}
```

68 シーケンスの最後の要素を除去する pop()

書式	void queue::pop();

●解説

空ではないシーケンスの最後の要素を除去します。

● 例

次の例は、stackに整数を保存して取り出す例です。

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <stack>

using namespace std;

int main(int argc, char *argv[])
{
   stack<int> stk;

   srand( (unsigned)time( NULL ) );

   // コンテナに値をプッシュする
   for(int i = 0 ; i < 5; ++i){
        stk.push( rand() % 100 );
   }

   // コンテナから値をポップする
   while( !stk.empty() ){
        cout << stk.top() << endl;
        stk.pop();
   }

   return 0;
}
```

69 シーケンスの最後の要素を除去する pop_back()

書式	① void list::pop_back(); ② void dequeue::pop_back(); ③ void vector::pop_back();

● 解説

空でないシーケンスの最後の要素を除去します。

● 例

次の例は、vectorの最後から要素を2個取り出すプログラムの例です。

```
#include <vector>
#include <iostream>

using namespace std;

int main(void)
{
   typedef vector<int>::iterator iterator;

   int d[] = {11, 8, 6, 4, 7, 3, 10};
   vector<int> v(d, d + 7);

   // コンテナの内容を出力する
   ostream_iterator<int,char> out(cout, ",");
   copy(v.begin(), v.end(), out);
   cout << endl;

   v.pop_back();
   v.pop_back();

   // 結果を出力する
   cout << "2個pop_back後:" << endl;
   copy(v.begin(), v.end(), out);
   cout << endl;

   return 0;
}
```

● 実行例

```
11,8,6,4,7,3,10,
2個pop_back後:
11,8,6,4,7,
```

70 シーケンスの最初の要素を除去する
pop_front()

書式	① void list::pop_front(); ② void dequeue::pop_front(); ③ void vector::pop_front();

●解説

空でないシーケンスの最初の要素を除去します。

●例… dequeの例参照。

●参照→pop_back()

71 ヒープから要素を取り出す
pop_heap()

ヘッダ	algorithm
書式	① template<class riter> void pop_heap(riter first, riter last); ② template<class riter, class pred> void pop_heap(riter first, riter last, pred pr);

●解説

このアルゴリズムは、ヒープから最も大きな要素を取り出します。

●参照→make_heap()

72 順序付け関数に従って連続した順列を生成する
prev_permutation()

書式	① template<class biter> bool prev_permutation(biter first, biter last); ② template<class biter, class pred> bool prev_permutation (biter first, biter last, pred pr);

●解説

このアルゴリズムは、順序付け関数に従って連続した順列を生成します。

●参照→next_permutation()

73 シーケンスの最後に要素を挿入する push()

書式	① void priority_queue::push(const T& x); ② void queue::push(const T& x); ③ void stack::push(const T& x);

● **解説**

シーケンスの最後に値xの要素を挿入します。

● **例**… stack、popの例参照。

74 シーケンスの最後に要素を挿入する push_back()

書式	① void deque::push_back(const T& x); ② void vector::push_back(const T& x); ③ void list::push_back(const T& x);

● **解説**

シーケンスの最後に値xの要素を挿入します。

● **例**… copy()、count_if()、erase()、fill_n()などの例参照

75 シーケンスの先頭に要素を挿入する push_front()

書式	① void deque::push_front(const T& x); ② void list::push_front(const T& x);

● **解説**

シーケンスの先頭に値xの要素を挿入します。

● **例**

次の例は、dequeの先頭に要素を追加する例です。

```
#include <deque>
#include <iostream>

using namespace std;

int main(void)
{
   deque<int> v;

   v.push_back(2);

   // コンテナの内容を出力する
   ostream_iterator<int,char> out(cout, ",");
   copy(v.begin(), v.end(), out);
   cout << endl;

   v.push_front(4);
   v.push_front(6);

   // 結果を出力する
   copy(v.begin(), v.end(), out);
   cout << endl;

   return 0;
}
```

● **実行例**

```
2,
6,4,2,
```

76 ヒープに値を保存する push_heap()

ヘッダ	algorithm
書式	① template<class riter> void push_heap(riter first, riter last); ② template<class riter, class pred> void push_heap(riter first, riter last, pred pr);

● **解説**

このアルゴリズムは、ヒープに新しい要素を入れます。

● **参照**→make_heap()

77 ストリームに文字を戻す putback()

ヘッダ	istream
書式	① istream::putback istream& istream::putback(char ch); ② basic_istream& basic_istream::putback(E c);

● 解説

直前に取り出された文字chをストリームに戻します。文字chはストリームから直前に取り出された文字でなければならず、そうでない場合の結果は保証されません。peek()を使えば、ストリームから文字を取り出さずに文字を調べることができます。

● 参照→peek()

78 要素の順番をランダムに変更する random_shuffle()

ヘッダ	algorithm
書式	① template<class riter> void random_shuffle(riter *first*, riter *last*); ② template<class riter, class Fun> void random_shuffle (riter *first*, riter *last*, Fun& *f*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *f* ………順序を決める関数。

● 解説

このアルゴリズムは、要素の順番をランダムに変更します。

● 例

次の例は、1から10までの値を持つvectorをランダムにシャッフルする例です。

```
#include <algorithm>
#include <vector>
#include <iostream>

using namespace std;

int main()
{
```

```
    vector<int> v;

    for (int i=1; i<11; i++)
        v.push_back(i);

    // 要素を出力する
    cout << "random_shuffle前:";
    copy( v.begin(), v.end(), ostream_iterator<int,char>(cout,",") );
    cout << endl;

    random_shuffle(v.begin(), v.end());

    // 要素を出力する
    cout << "random_shuffle後:";
    copy( v.begin(), v.end(), ostream_iterator<int,char>(cout,",") );
    cout << endl;

    return 0;
}
```

●実行例

```
random_shuffle前:1,2,3,4,5,6,7,8,9,10,
random_shuffle後:9,2,10,3,1,6,8,4,5,7,
```

79　逆シーケンスの先頭の位置を指す反復子を返す rbegin()

書式	① const_reverse_iterator Container::rbegin() const; ② reverse_iterator Container::rbegin();
戻り値	反復子

●解説

シーケンスの最後の直後の位置（逆シーケンスの先頭）を指す逆方向反復子を返します。

● 例

次の例は、vectorの要素を後ろから順に表示するプログラムの例です。

```
#include <vector>
#include <iostream>

using namespace std;

int main()
{
   vector<int> v;
   vector <int>::reverse_iterator rIter;


   for (int i=1; i<11; i++)
      v.push_back(i);

   for ( rIter = v.rbegin( ); rIter != v.rend( ); rIter++ )
      cout << *rIter << ",";
   cout << endl;


   return 0;
}
```

● 実行例

```
10,9,8,7,6,5,4,3,2,1,
```

80　ストリームからデータを取り出す read()

書式	① istream& istream::read(char* *pch*, int *n*); ② istream& istream::read(unsigned char* puch, int *n*); ③ istream& istream::read(signed char* psch, int n); ④ basic_istrem& basic_istream::read(E *s, streamsize *n*);
引数	*pch* ……取り出した文字列を保存するバッファのポインタ。 *n* ………取り出すデータの文字数。

● 解説

取り出した文字が*n*に達するかEOFに達するまで、ストリームから文字を取り出します。

● 例

次の例は、ファイルtest.txtの内容を出力するプログラムの例です。

```
#include <algorithm>
#include <fstream >
#include <iostream>

using namespace std;

int main()
{
   fstream fs("test.txt");

   char c;

   while ( !fs.eof() )
   {
      fs.read(&c, 1);
      cout << c;
   }

   return 0;
}
```

81 ストリームから文字を取り出す readsome()

書式	streamsize basic_istream::readsome(char_type* *s*, streamsize *n*);
引数	*s* ………取り出した文字列を入れるバッファのポインタ。 *n* ………取り出す文字数。

● 解説

ストリームから最大*n*文字を取り出して配列sに保存し、取り出した文字数を返します。

● 注意

ファイルからreadsome()で正しいデータを読み込めないバグがある場合があります。その場合は、readsome()の代わりにread()を使って問題を回避してください。

● 例

次の例は、ファイルからreadsome()で16バイトのデータを読み込むプログラムの例です。

```
#include <fstream>
#include <iostream>

using namespace std;

int main()
{
   fstream fs("test.txt");

   char str[20];
   str[16] = 0;

   fs.readsome(str, 16);
   // fs.read(str, 16);  Visual Studio 2005ではこちらを使う

   cout << str << endl;

   return 0;
}
```

82 要素を削除する remove()

ヘッダ	algorithm
書式	① template<class iter, class T> iter remove(iter *first*, iter *last*, const T& *val*); ② void list::remove(const T& *x*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *val*、*x* …削除する値。

● 解説

①のアルゴリズムは、条件（*iter == val）を満たす反復子iterによって表される要素をすべて削除します。コンテナから要素を実際に削除したい場合はerase()を使います。

②の関数はリストから要素を削除します。

●例

次の例は、vectorから値が5である要素を削除する例です。

```
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
   vector<int> v;
   vector <int>::iterator iter;

   for (int i=1; i<11; i++)
      v.push_back(i);

   for ( iter = v.begin(); iter != v.end(); iter++ )
      cout << *iter << ",";
   cout << endl;

   remove(v.begin(), v.end(), 5);

   for ( iter = v.begin(); iter != v.end(); iter++ )
      cout << *iter << ",";
   cout << endl;

   return 0;
}
```

●実行例

```
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,6,7,8,9,10,10,
```

83 要素を削除する remove_copy()

ヘッダ	algorithm
書式	template<class iiter, class oiter, class T> oiter remove_copy (iiter *first*, iiter *last*, oiter *x*, const T& *val*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の反復子。 *val*………削除する値。

● **解説**

このアルゴリズムは、要素を削除し、結果をxに返します。

● **参照**→remove()

84 要素を削除する remove_copy_if()

ヘッダ	algorithm
書式	template<class iiter, class oiter, class pred> oiter remove_copy_if(iiter first, iiter last, oiter x, pred pr);

● **解説**

このアルゴリズムは、条件prに一致する要素を削除します。

● **参照**→remove()

85 条件に一致する要素を削除する remove_if()

ヘッダ	algorithm
書式	① template<class iter, class pred> iter remove_if(iter first, iter last, pred pr); ② void list::remove_if(binder2nd<not_equal_to<T> > pr);

● **解説**

書式①のアルゴリズムは、条件prに一致する要素を削除します。コンテナから要素を実際に削除したい場合はerase()を使います。

書式②の関数は、条件prに一致する要素をすべてリストから削除します。

86	逆シーケンスの終端を指す反復子を返す **rend()**
書式	① const_reverse_iterator Container::rend() const; ② reverse_iterator Container::rend();

● **解説**

シーケンスの最初の要素または空であるシーケンスの直後の位置（逆シーケンスの終端）を指す逆方向反復子を返します。

● **参照**→rbegin()

87	要素を別の値で置き換える **replace()**
ヘッダ	algorithm
書式	template<class itor, class Type> void replace(iter *first*, iter *last*, const Type& *OldV*, const Type& *NewV*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *OldV*……置換される値。 *NewV* …置換する値。

● **解説**

このアルゴリズムは、要素を別の値で置き換えます。

● **例**

次の例は、vectorに3個の文字列を保存し、その中の“good”を“smart”に置き換える例です。

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   vector<string> v;
```

```
    // コンテナに値をプッシュする
    v.push_back( "dog" );
    v.push_back( "good" );
    v.push_back( "wanwan" );

    // コンテナの値を出力する
    for (int i=0; i< (int)v.size(); i++)
      cout << v[i] << endl;
    cout << endl;

    replace(v.begin(), v.end(), (const string)"good", (const string)"smart");

    // コンテナの値を出力する
    for (int i=0; i< (int)v.size(); i++)
      cout << v[i] << endl;

    return 0;
}
```

● 実行例

```
dog
good
wanwan

dog
smart
wanwan
```

88 要素を置き換え結果を別のコンテナに保存する replace_copy()

ヘッダ	algorithm
書式	template<class iiter, class oiter, class T> oiter replace_copy (iiter *first*, iiter *last*, oiter *x*, const T& *vold*, const T& *vnew*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………結果を保存するコンテナの反復子。 *vold* ……以前の値。 *vnew* …置き換えた値。

● **解説**

このアルゴリズムは、要素を別の値で置き換え、結果を別のコンテナにコピーします。

● **参照**→replace()

89 要素を置き換え結果を別のコンテナに保存する replace_copy_if()

ヘッダ	algorithm
書式	template<class iiter, class oiter, class pred, class T> oiter replace_copy_if(iiter *first*, iiter *last*, oiter *x*, pred *pr*, const T& *val*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………結果を保存するコンテナの反復子。 *pr* ………比較に使う述語。 *val*………置き換える値。

● **解説**

このアルゴリズムは、特定の条件prを満たす要素を別の値で置き換え、結果を別のコンテナにコピーします。

● **参照**→replace()、replace_copy()

90 要素を別の値で置き換える replace_if()

ヘッダ	algorithm
書式	template<class iter, class pred, class T> void replace_if(iter *first*, iter *last*, pred *pr*, const T& *val*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較に使う述語。 *val*………置き換える値。

● **解説**

このアルゴリズムは、要素を別の値で置き換えます。

● **参照**→replace()、replace_copy()、replace_copy_if()

91 iosフラグをリセットする resetiosflags

書式	resetiosflags(ios_base::fmtflags mask);

● 解説

このマニピュレータは、ストリームへの入出力の際のiosフラグをリセットします。

● 例… setiosflagsの例参照。

● 参照→setiosflags

92 コンテナのサイズを再設定する resize()

書式	① void Container::resize(size_type n); ② void basic_string::resize(size_type n);

● 解説

コンテナや文字列のサイズを再設定します。

● 例… fill()の例参照。

93 要素の順序を反転させる reverse()

ヘッダ	algorithm
書式	① template<class biter> void reverse(biter *first*, biter *last*); ② void list::reverse();
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。

● 解説

書式①のアルゴリズムと書式②の関数は、要素の順序を反転させます。

● 参照→reverse_copy()

94	要素の順序を反転させ結果を別のコンテナに保存する **reverse_copy()**
ヘッダ	algorithm
書式	template<class biter, class oiter> oiter reverse_copy(biter *first*, biter *last*, oiter *x*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の要素の反復子。

● 解説

このアルゴリズムは、要素の順序を反転させて、結果を別のコンテナにコピーします。

● 参照→reverse()

95	指定した範囲の値を残りの範囲の値と交換する **rotate()**
ヘッダ	algorithm
書式	template<class iter> void rotate(iter *first*, iter *middle*, iter *last*);
引数	*first* ……最初の要素の反復子。 *middle*…中央の要素の反復子。 *last* ……最後の要素の反復子。

● 解説

このアルゴリズムは、*first*から*middle*-1までの要素を、*middle*から*last*までの要素と交換します。

● 例

次の例は、5個の文字列を含むvectorの1～3個目の要素を、4と5番目の要素と交換します。

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <string>

using namespace std;

int main()
{
```

```
    vector<string> v;

    // コンテナに値をプッシュする
    v.push_back( "dog" );
    v.push_back( "good" );
    v.push_back( "happy" );
    v.push_back( "smart" );
    v.push_back( "wanwan" );

    // コンテナの値を出力する
    for (int i=0; i< (int)v.size(); i++)
      cout << v[i] << ",";
    cout << endl;

    rotate(v.begin(), v.begin()+3, v.end());

    // コンテナの値を出力する
    for (int i=0; i< (int)v.size(); i++)
      cout << v[i] << ",";
    cout << endl;

    return 0;
}
```

● **実行例**

```
dog,good,happy,smart,wanwan,
smart,wanwan,dog,good,happy,
```

● **参照**→rotate_copy()

96 指定した範囲の値を残りの範囲の値と交換し別のコンテナに保存する rotate_copy()

ヘッダ	algorithm
書式	template<class iter, class oiter> oiter rotate_copy(iter *first*, iter *middle*, iter *last*, oiter *x*);
引数	*first* ……最初の要素の反復子。 *middle* …中央の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の要素の反復子。

● **解説**

このアルゴリズムは、指定した範囲の値を残りの範囲の値と交換して、結果を別のコンテナにコピーします。

● **参照**→rotate()

97 指定した範囲で値が一致する範囲を検索する search()

ヘッダ	algorithm
書式	① template<class iter1, class iter2> iter1 search(iter1 *first1*, iter1 *last1*, iter2 *first2*, iter2 *last2*); ② template<class iter1, class iter2, class pred> iter1 search (iter1 *first1*, iter1 *last1*, iter2 *first2*, iter2 *last2*, pred *pr*);
引数	*first1*　…最初の要素の反復子。 *last1*……最後の要素の反復子。 *first2*　…検索する最初の要素の反復子。 *last2*……検索する最後の要素の反復子。 *pr*………比較に使う述語。

● **解説**

このアルゴリズムは、指定した範囲で値が一致する範囲を検索します。

● **例**

次の例は、文字リストls1の中からdogを探して、文字'd'を'D'に置き換える例です。

```
#include <string.h>
#include <algorithm>
#include <list>
#include <iostream>

using namespace std;

int main()
{
   char s1[] = "Very Good dog, Pochi";
   list<char> ls1(s1, s1+strlen(s1));

   char s2[] = "dog";
```

```
    list<char> ls2(s2, s2+strlen(s2));

    // dogを検索行する
    list<char>::iterator loc;
    loc = search(ls1.begin(), ls1.end(), ls2.begin(), ls2.end());

    // 文字'd'を'D'に置き換える
    *loc = 'D';

    // 結果を出力する
    list<char>::iterator iter;
    for(iter = ls1.begin(); iter != ls1.end(); iter++)
      cout << *iter;
    cout << endl;

    return 0;
}
```

98 指定した範囲で値が一致する範囲を検索する search_n()

ヘッダ	algorithm
書式	① template<class iter, class Dist, class T> iter search_n(iter *first*, iter *last*, Dist *n*, const T& *val*); ② template<class iter, class Dist, class T, class pred> iter search_n(iter *first*, iter *last*, Dist *n*, const T& *val*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *n* ………検索する長さ。 *val*………検索する値。 *pr* ………比較に使う述語。

●解説

このアルゴリズムは、指定した範囲で値が一致する範囲を検索します。

●例

次の例は文字列“Very Good dog, Pochi”をその内容とする文字のlistから、oが2個続く場所を検索して、それ以降を出力するプログラムの例です。

```
#include <list>
#include <iostream>
#include <algorithm>

using namespace std;

int main()
{
   char s[] = "Very Good dog, Pochi";
   list<char> ls(s, s+strlen(s));

   list<char>::iterator loc;
   loc = search_n(ls.begin(), ls.end(), 2, 'o');

   list<char>::iterator iter;
   for (iter=loc; iter != ls.end(); iter++ )
      cout << *iter;
   cout << endl;

   return 0;
}
```

●実行例

```
ood dog, Pochi
```

99 ストリームの入力ポインタを変更する seekg()

ヘッダ	istream
書式	① istream& istream::seekg(streampos *pos*); ② istream& istream::seekg(streamoff off, ios::seek_dir *dir*); ③ basic_istream& basic_istream::seekg(pos_type *pos*); ④ basic_istream& basic_istream::seekg(off_type off, ios_base::seek_dir *dir*);
引数	*pos* ……位置。 *dir*………シークする方向。

● 解説

入力ファイルストリームのファイルポインタの位置を設定します。ファイルポインタは、次に読み込むデータのファイル中の位置を指しています。

書式②と④の移動方向*dir*には、次の列挙子のいずれかを指定できます。

▼表 列挙子

列挙子	意味
ios::beg	ストリームの先頭から移動。
ios::cur	ストリームの現在位置から移動。
ios::end	ストリームの終わりから移動。

● 例

次の例は、ファイルを開き、ファイルの最後から16バイト前の位置に移動します。そして、バイトをファイルの最後まで読み込んでは標準出力に出力します。

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

#define FILENAME "./test.dat"

int main(int argc, char *argv[])
{
   char ch;
   ifstream ifs(FILENAME, ios::binary );
   if( !ifs )
```

```
    {
        cout << "ファイル" << FILENAME << "を開けません" << endl;
        return 1;
    }
    // 最後から16バイト前の位置に移動する
    ifs.seekg(-16, ios::end);
    // EOFかエラーが発生するまで読み込んで表示する
    while ( ifs.good() ) {
        ch = 0;
        ifs.get( ch );
        if (ch)
            cout << ch;
        else
            break;
    }
    return 0;
}
```

100 ソートされた差集合を作成する set_difference()

ヘッダ	algorithm
書式	① template<class iiter1, class iiter2, class oiter> oiter set_difference (iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*); ② template<class iiter1, class iiter2, class oiter, class pred> oiter set_difference(iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*, pred *pr*);
引数	*first1* …第1のコンテナの最初の要素の反復子。 *last1* ……第1のコンテナの最後の要素の反復子。 *first2* …第2のコンテナの最初の要素の反復子。 *last2* ……第2のコンテナの最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の要素の反復子。 *pr* ………比較に使う述語。

● 解説

2つのセットから、ソートされた差集合を作成します。

●例

次の例は2つのセットの差集合を出力します。

```
#include <algorithm>
#include <set>
#include <iostream>

using namespace std;

int main()
{
   //セットを初期化する
   int d1[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   set<int> set1(d1, d1+10);

   int a2[6]  = {3, 9, 12};
   set<int> set2(a2, a2+3);

   // set1を出力する
   ostream_iterator<int,char> out(cout, ",");
   copy(set1.begin(),set1.end(), out);
   cout << endl;

   copy(set2.begin(),set2.end(), out);
   cout << endl;

   set<int, less<int> > odd;
   insert_iterator<set<int, less<int> > > odd_ins(odd, odd.begin());
   set_difference(set1.begin(), set1.end(),
                  set2.begin(), set2.end(), odd_ins);

   copy(odd.begin(),odd.end(), out);
   cout << endl;

   return 0;
}
```

●実行例

```
1,2,3,4,5,6,7,8,9,10,
3,9,12,
1,2,4,5,6,7,8,10,
```

101 ソートされた交差集合を作成する set_intersection()

ヘッダ	algorithm
書式	① template<class iiter1, class iiter2, class oiter> oiter set_intersection (iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*); ② template<class iiter1, class iiter2, class oiter, class pred> oiter set_intersection(iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*, pred *pr*);
引数	*first1* …第1のコンテナの最初の要素の反復子。 *last1* ……第1のコンテナの最後の要素の反復子。 *first2* …第2のコンテナの最初の要素の反復子。 *last2* ……第2のコンテナの最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の要素の反復子。 *pr* ………比較に使う述語。

● 解説

2つのセットから、ソートされた交差集合を作成します。交差集合は、両方のセットのいずれにも含まれている要素からなる集合です。

● 例

次の例は、2つのvectorの交差集合を求める例です。

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
   vector < int > v1, v2, vr;
   vector < int >::iterator iter, vit;

   for (int i=0; i<10; i++){
      v1.push_back(i+1);
      v2.push_back(i+4);
      vr.push_back(0);
   }
   // コンテナの値を出力する
   cout << "v1:";
```

```
    for (int i=0; i< (int)v1.size(); i++)
      cout << v1[i] << ",";
    cout << endl;
    cout << "v2:";
    for (int i=0; i< (int)v2.size(); i++)
      cout << v2[i] << ",";
    cout << endl;

    vit = set_intersection(v1.begin(), v1.end(),
       v2.begin(), v2.end(), vr.begin());

    // 結果コンテナの値を出力する
    cout << "v1&v2(";
    for ( iter = vr.begin( ) ; iter != vit ; ++iter )
       cout << *iter << ",";
    cout << ")" << endl;

    return 0;
}
```

● **実行例**

```
v1:1,2,3,4,5,6,7,8,9,10,
v2:4,5,6,7,8,9,10,11,12,13,
v1&v2(4,5,6,7,8,9,10,)
```

102 ソートされた対称差集合を作成する set_symmetric_difference()

ヘッダ	algorithm
書式	① template<class iiter1, class iiter2, class oiter> oiter set_symmetric_difference(iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*); ② template<class iiter1, class iiter2, class oiter, class pred> oiter set_symmetric_difference(iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*, pred *pr*);
引数	*first1* …第1のコンテナの最初の要素の反復子。 *last1* ……第1のコンテナの最後の要素の反復子。 *first2* …第2のコンテナの最初の要素の反復子。 *last2* ……第2のコンテナの最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の要素の反復子。 *pr* ………比較に使う述語。

● **解説**

2つのセットから、ソートされた対称差を作成します。対称差集合は、両方のセットの一方に含まれている要素からなる集合です。

● **例**… set_intersection()の例参照。

103 ソートされた和集合を作成する set_union()

ヘッダ	algorithm
書式	① template<class iiter1, class iiter2, class oiter> oiter set_union (iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*); ② template<class iiter1, class iiter2, class oiter, class pred> oiter set_union(iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, iiter2 *last2*, oiter *x*, pred *pr*);
引数	*first1* …第1のコンテナの最初の要素の反復子。 *last1* ……第1のコンテナの最後の要素の反復子。 *first2* …第2のコンテナの最初の要素の反復子。 *last2* ……第2のコンテナの最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の要素の反復子。 *pr* ………比較に使う述語。

● **解説**

2つのセットから、ソートされた和集合を作成します。和集合は、両方のセットのいずれかに含まれている要素からなる集合です。

● **例**… set_intersection()の例参照。

104 ストリームへの入出力の際の基数を設定する setbase()

ヘッダ	iomanip
書式	setbase(int base);

● **解説**

このマニピュレータは、ストリームへの数値の入出力のときの基数を設定します。baseには、8、10、16のいずれかを指定します。これ以外の数値を指定したときには、ios_base::fmtflags(0)を指定したのと同じになります。

● **例**

次の例は、出力の基数を16に設定して、10進数で0から19までの値を出力する例です。

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
   cout << setbase(16);

   for (int i=0; i<20; i++)
   {
      int v = i;
      cout << v << " ";
   }
   cout << endl;

   return 0;
}
```

● 実行例

```
0 1 2 3 4 5 6 7 8 9 a b c d e f 10 11 12 13
```

105 ストリームへのパディング文字を設定する setfill()

ヘッダ	iomanip
書式	template<class E> setfill(E fillch);

● 解説

このマニピュレータは、ストリームへの入出力の際に、フィールドの長さに満たない部分に埋めるパディング文字を設定します。デフォルトのパディング文字は' '（半角の空白）です。

● 例… setw()の例参照。

106 iosフラグを設定する setiosflags()

ヘッダ	iomanip
書式	setiosflags(ios_base::fmtflags mask);

● 解説

このマニピュレータは、ストリームへの入出力の際のiosフラグを設定します。iosフラグは<ios>を参照してください。

● 例

次の例は、出力される整数値に+記号を表示するようにしたプログラムの例です。

```
#include <iostream>
#include <iomanip>

using namespace std;

int main(int argc, char *argv[])
{
   int v = 20;
```

```
    // 整数値に+記号を表示する
    cout << setiosflags(ios::showpos);
    cout << v << endl;

    // 整数値に+記号を表示するのをリセットする
    cout << resetiosflags(ios::showpos);
    cout << v << endl;

    return 0;
}
```

107 ストリームへの入出力の小数点以下の精度を設定する setprecision()

ヘッダ	iomanip
書式	setprecision(int prec);

●解説

このマニピュレータは、ストリームへの入出力の浮動小数点数の精度を設定します。

●例

次の例は、精度を3桁に設定して値を出力するプログラムの例です。

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    cout << setprecision(3);

    for (int i=1; i<11; i++)
    {
        double v = 3.1415 / (double)i;
        cout << v << endl;
    }

    return 0;
}
```

実行例

```
3.14
1.57
1.05
0.785
0.628
0.524
0.449
0.393
0.349
0.314
```

108 ストリームからの入出力のフィールド幅を設定する setw()

ヘッダ	iomanip
書式	setw(int wide);

解説

このマニピュレータは、ストリームからの入出力のフィールド幅を設定します。このマニピュレータは、ストリームの動作に対して一時的に影響を与えます。

例

次の例は、幅を指定して出力するプログラムの例です。

```
#include <iostream>
#include <iomanip>

using namespace std;

int main(int argc, char *argv[])
{
   int v = 20;
   cout << setfill('$');
   cout << setw(6) << v << endl; // 結果は"$$$$20"
   cout << v << endl; // 結果は"20"

   return 0;
}
```

109 シーケンスのサイズを返す size()

書式 ① size_type basic_string::size() const;
② size_t bitset::size() const;
③ size_type Container::size() const;

● **解説**… シーケンスの長さを返します。

● **例**… mapの例参照。

110 要素をソートする sort()

ヘッダ algorithm

書式 ① template<class riter> void sort(riter first, riter last);
② template<class riter, class pred> void sort(riter first, riter last, pred pr);
③ void list::sort();
④ template<class pred> void sort(greater<T> pr);

● **解説**

書式①と②のアルゴリズムと書式③以降の関数は、要素をソートします。

● **例**

次の例は、整数のvectorの要素を小さい順に並べ替えるプログラムの例です。

```
#include <algorithm>
#include <vector>
#include <iostream>
#include <functional>

using namespace std;

int main(void)
{
    typedef vector<int>::iterator iterator;

    int d[] = {11, 8, 6, 4, 7};
    vector<int> v(d, d + 5);
```

```
    // コンテナの内容を出力する
    ostream_iterator<int,char> out(cout, ",");
    cout << "before:" << endl;
    copy(v.begin(), v.end(), out);
    cout << endl;

    sort(v.begin(), v.end(), less_equal<int>());
    // 次のようにしても同じ
    // sort(v.begin(), v.end());

    // 結果を出力する
    cout << "after:" << endl;
    copy(v.begin(), v.end(), out);
    cout << endl;

    return 0;
}
```

111 ヒープをソートされたコレクションに変換する sort_heap()

ヘッダ	algorithm
書式	① template<class riter> void sort_heap(riter *first*, riter *last*); ② template<class riter, class pred> void sort_heap(riter *first*, riter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較に使う述語。

● 解説

このアルゴリズムは、ヒープをソートされたコレクションに変換します。

● 参照→make_heap()

112 条件を満足する要素をすべて満足しない要素より前に配置する stable_partition()

ヘッダ	algorithm
書式	template<class iter, class pred> iter stable_partition(iter *first*, iter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較に使う述語。

● 解説

このアルゴリズムは、条件を満足する要素をすべて満足しない要素より前に配置します。

113 要素をソートする stable_sort()

ヘッダ	algorithm
書式	① template<class riter> void stable_sort(riter *first*, riter *last*); ② template<class riter, class pred> void stable_sort(riter *first*, riter *last*, pred *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較の述語。

● 解説

このアルゴリズムは、要素をソートします。等しい要素の相対的な順序は保たれます。

● 例

次の例は、vectorに20個の整数値を保存してソートする例です。

```
#include <stdlib.h>
#include <time.h>
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
```

```
    vector<int> v;

    srand( (unsigned)time( NULL ) );

    // コンテナに値をプッシュする
    for(int i = 0 ; i < 20; ++i)
        v.push_back( rand() % 100 );

    // コンテナの内容を出力する
    cout << "v=";
    copy(v.begin(), v.end(), ostream_iterator<int,char>(cout,","));
    cout << endl;

    // ソートする
    stable_sort(v.begin(), v.end());

    cout << "v=";
    copy(v.begin(), v.end(), ostream_iterator<int,char>(cout,","));
    cout << endl;

    return 0;
}
```

● **実行例**

```
v=56,7,66,85,7,4,81,21,80,64,8,13,97,32,10,42,38,30,79,52,
v=4,7,7,8,10,13,21,30,32,38,42,52,56,64,66,79,80,81,85,97,
```

114 コンテナ全体の値を交換する swap()

ヘッダ algorithm

書式
① template<class Key, class T, class pred, class A> void swap (const map <Key, T, pred, A>& lhs, const map <Key, T, pred, A>& rhs);
② template<class Key, class T, class pred, class A> void swap (const multimap <Key, T, pred, A>& lhs, const multimap <Key, T, pred, A>& rhs);
③ template<class T, class A> void swap(const list <T, A>& lhs, const list <T, A>& rhs);
④ void basic_string::swap(basic_string& str);
⑤ void container::swap(container& str);

● 解説

アルゴリズムswap()および関数swap()は、コンテナまたはオブジェクト全体の値を交換します。

● 例

次の例は、リストの内容をそっくり交換する例です。

```
#include <list>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
   list<int> l1, l2;

   int i;
   for (i=0; i<5; i++)
      l1.push_back(i);
   for (i=0; i<8; i++)
      l2.push_back(10 - i);

   // コンテナの内容を出力する
   cout << "l1=";
   copy(l1.begin(), l1.end(), ostream_iterator<int,char>(cout,","));
   cout << endl;
```

```
    // コンテナの内容を出力する
    cout << "l2=";
    copy(l2.begin(), l2.end(), ostream_iterator<int,char>(cout,","));
    cout << endl << endl;

    // リストを入れ換える
    swap(l1, l2);

    // コンテナの内容を出力する
    cout << "l1=";
    copy(l1.begin(), l1.end(), ostream_iterator<int,char>(cout,","));
    cout << endl;
    // コンテナの内容を出力する
    cout << "l2=";
    copy(l2.begin(), l2.end(), ostream_iterator<int,char>(cout,","));
    cout << endl << endl;

    return 0;
}
```

● **実行例**

```
l1=0,1,2,3,4,
l2=10,9,8,7,6,5,4,3,

l1=10,9,8,7,6,5,4,3,
l2=0,1,2,3,4,
```

115 コンテナの指定した範囲の値を交換する swap_ranges()

ヘッダ	algorithm
書式	template<class ForwardIterator1, class ForwardIterator2> itor2 swap_ranges(itor *first1*, itor *last1*, itor *first2*);
引数	*first1* …最初の要素の反復子。 *last1* ……最後の要素の反復子。 *first2* …交換する最初の要素の反復子。

● **解説**

このアルゴリズムは、コンテナの指定した範囲の値を交換します。

●例

次の例は、10個の要素を持つvectorの最初の3個の要素と最後の5個の要素のうち3個を入れ換える例です。

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
   int d1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   vector<int> v(d1+0,d1 + 10);

   // コンテナの内容を出力する
   copy(v.begin(), v.end(), ostream_iterator<int,char>(cout,","));
   cout << endl;

   // 最初の3個の要素と最後の5個の要素のうち3個を入れ換える
   swap_ranges(v.begin(), v.begin() + 3, v.begin() + 5 );

   // 結果を出力する
   copy(v.begin(), v.end(), ostream_iterator<int,char>(cout,","));
   cout << endl;

   return 0;
}
```

●実行例

```
1,2,3,4,5,6,7,8,9,10,
6,7,8,4,5,1,2,3,9,10,
```

116 ストリームバッファと外部から入力される文字との同期をとる sync()

ヘッダ	fstream
書式	① int istream::sync(); ② int basic_istream::sync(); ③ int basic_filebuf::sync() ④ virtual int basic_streambuf::sync(); ⑤ virtual int streambuf::sync();

解説

ストリームの内部バッファと外部からの文字入力の同期をとり、出力をフラッシュします。エラーが発生したときはEOFを返します。

117 ストリームの入力ポインタの値を取得する tellg()

ヘッダ	istream
書式	① streampos istream::tellg(); ② pos_type basic_istream::tellg();

解説

ストリームの入力ポインタの位置の値を取り出します。

例

次の例は、ファイルを開いてファイルの最後にファイルポインタを移動し、その位置を取得して表示します。結果として、ファイルのサイズがわかります。

```
#include <iostream>
#include <fstream>

using namespace std;

#define FILENAME "./test.dat"

int main(int argc, char *argv[])
{
   ifstream ifs(FILENAME, ios::binary );

   if( !ifs )
```

```
    {
        cout << "ファイル" << FILENAME << "を開けません" << endl;
        return 1;
    }

    // 最後に移動する
    ifs.seekg(0, ios::end);

    // ストリームのポインタの位置を取得して表示する
    long pos = ifs.tellg();
    cout << pos << endl;

    return 0;
}
```

118 文字を小文字に変換する tolower()

ヘッダ	locale
書式	template <class charT> charT ctype::tolower (charT *c*, const locale& *loc*) const;
引数	*c* ………文字。 *loc*………ロケール。

● 解説

ロケール*loc*に従って文字*c*を小文字に変換して返します。

● 例

次の例は、ロケール"C"に従って文字cを大文字に、文字Fを小文字に変換して表示します。

```
#include <iostream>
#include <locale>

using namespace std;

int main(int argc, char *argv[])
{
    locale loc = locale("C");
    cout << 'a' << (toupper('c', loc)) << (tolower('F', loc)) << endl;
    return 0;
}
```

119 文字を大文字に変換する toupper()

ヘッダ	locale
書式	template <class charT> charT ctype::toupper (charT *c*, const locale& *loc*) const;
引数	*c* ………文字。 *loc*………ロケール。

● 解説

ロケール*loc*に従って文字*c*を大文字に変換して返します。

● 例

tolower()の例参照。

120 指定した範囲の値に演算を適用する transform()

ヘッダ	algorithm
書式	① template<class iiter, class oiter, class Unop> oiter transform(iiter first, iiter last, oiter *x*, Unop *uop*); ② template<class iiter1, class iiter2, class oiter, class Binop> oiter transform(iiter1 *first1*, iiter1 *last1*, iiter2 *first2*, oiter *x*, Binop *bop*); ③ string_type collate::transform(const E *first, const E *last) const;
引数	*first1* ……………最初の要素の反復子。 *last1* ……………最後の要素の反復子。 *first2* ……………ソースの第2要素の反復子。 *x* …………………結果を保存する最初の要素の反復子。 *uop*、*bop* ………要素に適用する操作。

● 解説

書式①と②のアルゴリズムと書式③の関数は、指定した範囲の値に演算を適用します。

04-03 標準C++のクラスのメンバとアルゴリズム

●例

次の例は、値を2倍にする演算をvectorの各要素に適用するプログラムの例です。

```
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int uop(int n) // 2倍する演算関数
{
   return n * 2;
}

int main(int argc, char *argv[])
{
   vector < int > v1, v2, vr;
   vector < int >::iterator iter, vit;

   for (int i=0; i<10; i++){
      v1.push_back(i+1);
      vr.push_back(0);
   }

   // コンテナの値を出力する
   cout << "v1:";
   for ( iter = v1.begin( ) ; iter < v1.end() ; ++iter )
      cout << *iter << ",";
   cout << endl;

   vit = transform(v1.begin(), v1.end(), vr.begin(), uop);

   // 結果コンテナの値を出力する
   cout << "vr:";
   for ( iter = vr.begin( ) ; iter != vit ; ++iter )
      cout << *iter << ",";
   cout << endl;

   return 0;
}
```

● **実行例**

```
v1:1,2,3,4,5,6,7,8,9,10,
vr:2,4,6,8,10,12,14,16,18,20,
```

121 重複している要素を削除する unique()

ヘッダ	algorithm
書式	① template<class iter> iter unique(iter *first*, iter *last*); ② template<class iter, class pred> iter unique(iter *first*, iter *last*, pred *pr*); ③ void list::unique(); ④ void list::unique(not_equal_to<T> *pr*);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *pr* ………比較に使う述語。

● **解説**

このアルゴリズムと関数は、コンテナから重複している要素を削除します。

● **例**

次の例は、vectorに20個の整数値を保存し、重複している要素を削除する例です。

```
#include <stdlib.h>
#include <time.h>
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
   vector<int> v;
   vector<int>::iterator iter, EndIter;

   srand( (unsigned)time( NULL ) );
```

```
    // コンテナに値をプッシュする
    for(int i = 0 ; i < 20; ++i)
        v.push_back( rand() % 10 );

    sort( v.begin(), v.end() );

    // コンテナの値を出力する
    for (int i=0; i< (int)v.size(); i++)
      cout << v[i] << ",";
    cout << endl;

    EndIter = unique( v.begin(), v.end() );

    // コンテナの値を出力する
    for (iter = v.begin(); iter<EndIter; iter++)
      cout << (*iter) << ",";
    cout << endl;

    return 0;
}
```

● 実行例

```
1,1,1,1,2,3,3,3,4,4,4,4,5,6,6,6,8,8,9,9,
1,2,3,4,5,6,8,9,
```

● 参照→unique_copy()

122 重複している要素を削除し、別のコンテナに保存する unique_copy()

ヘッダ	algorithm
書式	① template<class iiter, class oiter> oiter unique_copy(iiter *first*, iiter *last*, oiter *x*); ② template<class iiter, class oiter, class pred> oiter unique_copy(iiter *first*, iiter *last*, oiter *x*, pred pr);
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *x* ………結果を保存するコンテナの最初の要素の反復子。

● **解説**

このアルゴリズムは、重複している要素を削除し、重複している要素を別のコンテナにコピーします。

123 コンテナの値が入る上限を示す反復子を返す upper_bound()

ヘッダ	algorithm
書式	① template<class iter, class T> iter upper_bound(iter *first*, iter *last*, const T& *val*); ② template<class iter, class T, class pred> iter upper_bound(iter *first*, iter *last*, const T& *val*, pred *pr*); ③ iterator AssoCont::upper_bound(const Key& key); ④ const_iterator AssoCont::upper_bound(const Key& key) const;
引数	*first* ……最初の要素の反復子。 *last* ……最後の要素の反復子。 *val*………最小の要素の値。 *pr* ………比較に使う述語。

● **解説**

書式①と書式②のアルゴリズムは、ソート済みのコンテナで、値が入る上限を示す反復子を返します。

書式③以降の関数は、連想コンテナで`key_comp()(key, x.first)`を真とするような最初の要素xを指す反復子を返します。

124 要素の順序を決定する関数オブジェクトを返す value_comp()

書式	value_compare AssoCont::value_comp() const;

● **解説**… 連想コンテナの要素の順序を決定する関数オブジェクトを返します。

125 バッファからストリームにバイト列を出力する write()

ヘッダ	ostream
書式	① basic_ostream& basic_ostream::write(const E *s, streamsize *n*); ostream& ostream::write(const char* *pch*, int *n*); ② ostream& ostream::write(const unsigned char* puch, int *n*); ③ ostream& ostream::write(const signed char* psch, int *n*);
引数	*pch* ……出力する文字列のポインタ。 *n* ………出力する文字数。

● 解説

指定されたバイト数nをバッファからストリームに出力します。

write()は、通常、バイナリストリームへの出力に使いますが、ファイルがテキストモードで開かれているときは、出力に改行文字が付加されます。

● 例

次の例は、“test.dat”という名前のファイルに、“Hello, good dogs.”という文字列を書き込むプログラムの例です。

```
#include <fstream >
#include <iostream>

using namespace std;

int main()
{
   fstream os;

   os.open("test.dat", ios_base::out);

   char str[] = "Hello, good dogs.";

   os.write(str, strlen(str));

   os.close();

   return 0;
}
```

126 先行する空白文字を取り除く
ws

書式	template class<E, T> basic_istream<E, T>& ws(basic_istream <E, T> is);

●解説

このマニピュレータは、スペースとみなされる要素を入力ストリームから取り出して破棄します。ストリームから空白文字を取り除くことができます。

●例

次の例は、先行する空白を除いて標準入力から文字列を入力する例です。

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   char str[80];

   // 先行するホワイトスペースを取り除く
   cin >> ws;

   // 80文字か改行文字まで読み取る
   cin.getline( str, 80 );

   cout << "[" << str << "]" << endl;

   return 0;
}
```

●実行例

```
        abcdef ghij
[abcdef ghij]
```

05: そのほかのライブラリ

ここでは、X Window System、Windows、OpenGLなど、C/C++のプログラミングで特に重要なライブラリの概要を説明します。

X Window System

UNIX系の多くのシステムでは、X Window System（以下、Xと略）を利用します。LinuxやFreeBSDのようないわゆるPC-UNIXでは、フリーのXFree86を使いますが、XとXFree86は実質的に同じです。

Xはクライアント/サーバーモデルを使ってハードウェアに依存しないユーザーインタフェイスを提供するイベント駆動型のグラフィカルなウィンドウシステムです。

Xのプログラミングには、C/C++言語の標準ライブラリのほかに、さまざまなライブラリやツールキットの中から、自分の目的に適したものを選択して利用します。以下に主なライブラリとツールキットの概要を示します。いずれのツールキットも、初期のバージョンでは日本語をそのままでは扱えませんでしたが、現在では内部でUnicodeを使って国際化に対応することが一般的になり、日本語環境でも問題なく使えるようになってきました。

Xlib

XlibはXのプリミティブなC言語ベースの関数ライブラリで、ウィンドウへの描画、仮想カラーマップ、イベントの処理など、基本的で必要不可欠なAPIを提供します。Xlibは低レベルの機能だけを提供するライブラリなので、メニューやボタンなどのさまざまなGUIオブジェクトを使うXプログラミングでは、より高レベルのオブジェクトや機能を提供するほかのウィジェットやツールキットを併用します。

Athenaウィジェット

Xプログラミングでは、ボタンやチェックボックスのような部品として利用できるオブジェクトを、ウィジェット（Widget）と呼びます。Athenaウィジェットは初期からあるウィジェットの1つです。最近では、拡張されたXaw3d（3DAthena ウィジェット）ライブラリも使われています。

Motif

OSF（The Open Software Foundation）が提唱しているXプログラミングのためのツールキットの1つで、オブジェクト指向のウィジェットを利用してXクライアントの外観と操作性（Look & Feel）を実装できるようにします。商用であるSunのOpen Lookとともに、さまざまな面で、以前のXプログラミングの標準をになっていました。

GTKとGTK+

GTK（GIMP Tool Kit）はGIMPというグラフィックスアプリケーションを開発するために作られたツールキットですが、現在ではさまざまなXプログラミングで利用されています。

GTK+はオープンソースのフリーソフトウェアとして配布されている、Xのプログラム開発のためのオブジェクト指向のGUIツールキットです。GTK+のC++インターフェイスとして、Gtk--もあります。

V

VはXプログラミングのためのオブジェクト指向のC++クラスのツールキットです。これはGNU General Public Licenseに従って配布されているので、ライセンスの制限内で無料で自由に利用することができます。VはXaw3d、OpenGL、MotifなどのXlibツールキットも併用することができます。また、Vを使うと、Microsoft WindowsとLinuxそのほかを含むUNIXでソースコード互換のアプリケーションを作成できます。

Qt

QtはTroll Techが開発した、C++をベースとしたXのプログラミング用のGUIツールキットです。Qtは基本的にはTroll Techの商品ですが、Unix/X版のソースコードは公開されていて、フリーソフトウェア開発には無料で利用することができます。

Microsoft Windows

Microsoft Windows（以下、Windowsと略）はMicrosoftが提供するイベント駆動型のグラフィカルなウィンドウシステムです。Windowsのプログラミングには、C/C++言語の標準ライブラリのほかに、Windows APIと、MFCやそのほかのクラスライブラリを利用します。

Windows API

Windowsアプリケーションやその他のプログラムが、Windowsに対してファイルシステムへのアクセスやGUIへの出力のような低水準サービスを必要とするときに使うインターフェイスのことをWindows APIと呼びます。

APIはApplication Programming Interface（アプリケーションプログラミングインターフェイス）の略です。伝統的なWindowsのプログラミングでは、API関数を使いました。API関数は、アプリケーションやコンポーネントがWindowsのインターフェイスにアクセスするために使う一連の関数のことです。API関数はWindowsと共に提供されるさまざまなダイナミックリンクライブラリ（DLL：Dynamic Link Library）に含まれています。

アプリケーションのウィンドウやメニュー、ダイアログボックスなどはAPIで一元管理されていて、プログラムから要求されるとAPI ライブラリの関数が必要な操作を行います。

.NET Framework

主にインターネットをはじめとするネットワークでの利用を重視した新しいWindowsプラットフォームである、.NET Platformのためのフレームワークです。これはCLR（Common Language Runtime）を利用するプログラミング言語が共通して利用可能な専用のクラスライブラリを含む実行環境で、プログラムはこのライブラリを通して.NETフレームワークが提供する機能にアクセスします。.NETのプログラムは、C++/CLI Managed C++、C#、VisualBasic.NETなどを使って開発することができ、MSIL（Microsoft Intermediate Language）にコンパイルされてCLRで実行されます。

Direct X

Microsoftが開発した、主にゲームやマルチメディアをターゲットとしたソフトウェア技術の総称。DirectXの技術の中には、高度なグラフィックス処理を高速で行う「Direct3D」、「D3DX」、「DirectDraw」、3Dサウンドを含むサウンドの技術「DirectSound」、さまざまな入力デバイスを利用できるようにする「DirectInput」、ネットワーク処理を行なう

「DirectPlay」、既存の動画フォーマットのファイル（MPEGなど）を処理する「DirectShow」など、複数の技術が含まれます。

DirectX 9ではC# やVisual Basic、.NETから利用可能なManaged DirectXが登場しました。

http://www.microsoft.com/japan/msdn/directx/

WindowsのDLL

WindowsのDLLは、Windows環境で実行時にプログラムと動的にリンクされるライブラリモジュールです。WindowsにはさまざまなDLLが用意されていますが、自分でDLLを作成することもできます。

C言語でDLLを作るときには、少なくとも次のようなファイルを用意します。

1）関数を書いたC言語のソースファイル
2）その関数を利用できるようにするためのDEFファイル

関数を書いたC言語のソースファイルでは、ファイルWindows.hを必ずインクルード（#include）します。また、Visual Basicそのほかの言語から呼び出せるようにするために関数名の前に__stdcallを付けます。

WindowsのDLLのC言語のソースリストの例をリスト5.1に示します。

▼ リスト5.1　CalcSw.c

```
//
// CalcSw.c
//
#include <Windows.h>

double __stdcall fnCalcSW(int h, int sex)
{
   double h1;
   h1 = h * 0.01;

   return h1 * h1 * (23 - sex);
}
```

05-02　Microsoft Windows

DLLとその関数を利用できるようにするためのDEFファイルは、LIBRARY文にDLLの名前を指定し、DLLの外から呼び出せるようにしたい関数名と番号（@1、@2...）をEXPORTSのあとに列記します。DEFファイルの例をリスト5.2に示します。

▼ リスト5.2　CalcSW.DEF

```
; CalcSW.DEF
LIBRARY CalcSW
EXPORTS
fnCalcSW @1
```

WindowsのDLLをビルドするときには、Windowsで使えるC言語のコンパイラとWindows SDKが必要です。

OpenGL

OpenGLは、SGI（Silicon Graphics, Inc.）が開発した3Dグラフィックスライブラリをベースにして、実行時にリアルタイムで3次元グラフィックスを生成して効率的に表示するために開発された、3Dグラフィックスプログラミングインターフェイスです。

OpenGLは、基本的にはOSに依存しない3次元グラフィックスライブラリ（API）です。そのため、現在では、Windows、UNIX系OS（Linux、Solaris、FreeBSDなど）、Macintosh（Mac OS X）など、ほとんどの主要なOSにはOpenGLのライブラリが用意されています。そして、OpenGLが利用できる環境であれば、どの実行環境であっても3Dグラフィックスをほぼ同じように表示できます。

OpenGLのプログラムは、OpenGLのライブラリを直接使って開発することもできます。しかし、GLUT（The OpenGL Utility Toolkit）という名前のツールキットを利用すると、OpenGLのプログラミングをより容易に行うことができます。また、C++で作られているGLUTベースのユーザーインタフェイスライブラリであるGLUI（OpenGL User Interface Library）を利用することで、OpenGLのプログラムをさらに容易に開発することができます。GLUIは、OpenGLアプリケーションで利用できる、ボタン、チェックボックス、ラジオボタンなどのコントロールを提供するためのユーザーインターフェイスライブラリです。

OpenGLとその関連情報は更新されているので、最新情報をWebサイトで参照することをお勧めします。たとえば、以下のようなサイトがあります。

http://www.opengl.org/
http://developer.apple.com/opengl/
http://www.opengl.org/developers/documentation/specs/

付録A：C/C++プログラミングのファイル

ここではC/C++プログラミングに関連するファイルを、コマンド、ファイル、標準的なファイル拡張子に分けて説明します。

ファイルとファイル拡張子

ここでは、C/C++プログラム開発で頻繁に使うファイルと、ファイル拡張子の概要を示します。ファイル拡張子はファイルの最後の.（ピリオド）以降の部分です。

主なファイル

main.c

C言語プログラムのメインソースモジュール。プログラムのメインソースモジュールに付ける名前に決まりはありませんが、特別な理由がない限り、main.c、プログラム名.c、コマンド名.cなど、ほかのプログラマにとってもわかりやすい名前にするべきです。

main.cpp、main.cc、main.cxx

C++プログラムのメインソースモジュール。プログラムのメインソースモジュールに付ける名前に決まりはありませんが、特別な理由がない限り、main.cpp、プログラム名.cpp、コマンド名.cpp、あるいはmain.cc、プログラム名.cc、コマンド名.cxxのような、ほかのプログラマにとってもわかりやすい名前にするべきです。

makefile

プログラムをコンパイルするときに使うファイル。このファイルには、プログラムを構成するファイルとその依存関係を記述し、プログラムを効率的に更新できるようにします。また、コンパイル以外のファイル処理や操作にも使うことができます。

ファイル名の先頭を大文字にしたMakefileというファイル名でもかまいません。

makefileの基本的な構造は次のとおりです。

```
（生成するファイル名）　：　（ソースとなるファイルのリスト）
＜タブ＞（ファイルを生成するためのコマンド）
```

最も単純なmakefileの例をリストA-1に示します。これは、UNIX系OSでC言語のソースプログラムファイルhello.cから、実行可能ファイルhelloを作成するためのmakefileの例です。

リストA-1　helloプログラムのためのmakefile

```
hello : hello.c
        gcc -o hello hello.c
```

マクロを使った単純なmakefileの例をリストA-2に示します。これは、C言語のソースプログラムファイルmain.cから、実行可能ファイルmyprogを作成するためのmakefileの例です。

リストA-2　makefileの例

```
PROGRAM = myprog
SRC = main.c

all:$(PROGRAM)

$(PROGRAM):$(SRC)
        gcc -o $(PROGRAM) $(SRC)

install:
        cp ./$(PROGRAM) /usr/local/bin/
```

`all:$(PROGRAM)`の行は省略することもできます。省略しない場合、**make all**でプログラムをコンパイルして、**make install**でファイルをコピー（**cp**）することができます。

A-01　ファイルとファイル拡張子

主な拡張子

C/C++プログラミングで使うファイルは、通常、ファイルの種類に従って一定の拡張子が付けられます。以下に、一般的に使われる拡張子を示します（拡張子の解釈はコンパイラの種類やオプション指定などで変わることがあります)。

▼ 主な拡張子

拡張子	意味
.a	アーカイブファイル（リンカがリンクするファイル）
.C	C言語ソースファイル
.c	C言語ソースファイル
.cc	C++言語ソースファイル
.cpp	C++言語ソースファイル
.cxx	C++言語ソースファイル
.h	C/C++言語のヘッダファイル
.i	プリプロセッサで処理されたC言語の中間ファイル
.ii	プリプロセッサで処理されたC++言語の中間ファイル
.lib	ライブラリファイル（Windows）
.m	Objective-Cのソースファイル
.o	オブジェクトファイル
.obj	コンパイル後のオブジェクトファイル（Windows）
.S	アセンブラソースファイル
.s	アセンブラソースファイル

主なコマンドとプログラム

ここでは、C/C++プログラム開発で最も頻繁に使うコマンドおよびプログラムの概要を示します。

ここに示すコマンドは基本的にはUNIX系のOS上のコマンドですが、ほとんどのコマンドはWindowsをはじめとするほかの環境でも用意されています。

emacsやmuleはUNIX系のOSのアプリケーションです。Windowsでは、一般的には、Visual StudioやBorland Developer Studioのような統合開発環境（IDE）がよく利用されています。UNIX系のOSのIDEではeclipseがよく使われます。

01　高機能エディタプログラム emacs/mule

書式	emacs [command-line switches] [files...]

● 機能

C/C++プログラム開発に適したエディタプログラムです。

● 解説

emacs、xemacs、muleはRichard Stallmanが開発したemacsエディタをベースにして作られたテキスト編集用プログラムです。emacsを日本語などマルチバイト文字に対応させたプログラムがmule、X Window System（以下、Xと略）のアプリケーションとして作成されたemacsがxemacsですが、バージョンによってはコマンドとして**emacs**を入力することでX上でも利用でき、日本語も編集可能なプログラムとして作られています。xemacs、muleなど、emacsから派生したエディタをemacs系エディタといいます。

以下に示す操作はC/C++プログラミングでもっとも頻繁に使う操作です。これらの操作方法を活用することでemacs系エディタの中で開発やデバッグを行うことができます。

なお、以下の操作方法の説明において、**[M]**はメタキーの略で、PCの標準的なキーボードでは**[Alt]**キーに相当します。たとえば、**[M]-[x]compile**はPCの標準的なキーボードでは**[Alt]+[x] compile**です。**[C]**は**[Ctrl]**キーです。

● ヘルプ… [C]-[h]

操作方法を表示します。

● 保存… [C]-[x] [C]-[s]

現在編集中のテキストをディスクに保存します。

● **終了… [C]-[x] [C]-[c]**

エディタを終了します。

● **コンパイル… [M]-[x] compile**

[M]+[x]compileで**make -k**を実行します。この機能を利用するためには、makefileを作っておく必要があります。

● **実行… [M]-[!]**

プログラムを実行します。

● **エラー個所へのジャンプ… [C]-[x]**

コンパイルエラーが発生したら、**[C]-[x]**でエラーの場所にジャンプします。

02	GNUプロジェクトC/C++コンパイラ gcc、g++
書式	gcc [option	filename]... g++ [option	filename]...

● **機能**

C/C++ソースプログラムをコンパイルします。

● **解説**

最近のGNUコンパイラは、CとC++コンパイラが統合されています。どちらも、プリプロセス、コンパイル、アセンブル、リンクという4ステップでソースファイルをコンパイルします。デフォルトでは、gccはC言語のソースプログラムを仮定し、プリプロセッサで処理されたファイル（.iファイル）がCであると想定して、C言語スタイルのリンクを仮定します。g++は、C++のソースプログラムを仮定し、プリプロセッサで処理されたファイル（.iファイル）がC++であると想定して、C++スタイルのリンクを仮定します。

gccの主なコマンドラインオプションは次のとおりです（コマンドラインオプションはgccのバージョンによって多少異なることがあります）。

▼表　gccの主なコマンドラインオプション

オプション	機能
-c	リンクしないで、コンパイルだけを行う。
-Dmacro	macroを定数として定義する。macro=defineの形式で、値や式などをマクロとして定義することもできる。
-g	デバッガで利用できるデバッグ情報を生成する。
-S	コンパイルして拡張子が.sのアセンブラファイルを出力する。
-E	プリプロセスだけ行い、標準出力に出力する。
-ofile	出力ファイルを指定する。このオプションで出力ファイル名を指定しなければ、デフォルトの実行可能ファイルa.outに出力される。
-On	最適化のレベルを指定する。nは1または2を指定するか省略可能。
-P	C言語プリプロセスの結果を出力する。
-v	コンパイラのバージョンやコンパイル過程などを標準出力で表示する。
-Idir	インクルードファイルの検索ディレクトリパスを指定する。
-Ldir	ライブラリファイルの検索ディレクトリパスを指定する。

03　プログラムのビルドとファイル処理
make

書式　make [-f makefile][option] ... target ...

● 機能

プログラムをビルドしたり、ファイルを処理します。

● 解説

makeはmakefile（またはMakefileかオプションで-f指定したファイル）の情報に従って、ソースプログラムから実行可能プログラムファイルを作成したり、そのほかのファイル処理や操作を行います。

● 参照… makefile（486ページ）

04 プログラムのデバッグ gdb

書式	gdb [-f makefile][option] ... target ...

● 機能

プログラムをデバッグします。

● 解説

gdbはプログラムを実行しながらバグを探すときに使います。デバッグのために、条件を指定してプログラムを停止させたり、プログラムが停止したときの状態を調べることができます。また、プログラムの状態を変更して、その影響を調べることもできます。

gdbにはとてもたくさんのコマンドがありますが、主なコマンドを以下に示します。

▼表　gdbの主なコマンド

コマンド	機能
break [file:]function	ブレークポイントをファイルfileにある関数functionに設定する。
run [arglist]	プログラムの実行を開始する。必要に応じて引数リストarglistを指定することも可能。
bt	バックトレースする。プログラムのスタックを表示する。
print expr	式exprの値を表示する。
c	ブレークポイントで停止中にプログラムの実行を再開する。
next	停止中に次のプログラム行を実行する。関数を呼び出している場合、呼び出している関数全体を実行する。
step	停止中に次のプログラム行を実行する。関数を呼び出している場合、呼び出している関数の中に入って次の行を実行する。
help [name]	gdbの説明またはコマンドnameに関する情報を表示する。
quit	gdbを終了する。

付録B：トラブル対策

ここではC/C++プログラミングで一般的によく発生する可能性があるトラブルとその対策を示します。

一般的なトラブルとその対処

トラブル①　ccやgcc、makeなどを実行できない

・開発ツールがインストールされていないと、実行できません。通常、UNIX系のC/C++コンパイラの名前は、cc、gcc、g++ですが、異なる名前のコンパイラがインストールされている場合もあります。

・開発ツールがインストールされていても、パスが通っていないと実行できません。これらのツールが正しくインストールされているかどうか調べるためには、実際に小さなプログラムを作成してコンパイルしてみるのがいちばん良い方法です。また、コマンドに引数**--version**を付けて実行してみるのも良いでしょう。

--versionはたいていのコマンドで、そのコマンドのバージョンを表示します。

```
# gcc --version
2.7.2.3
# make --version
GNU Make version 3.76.1, by Richard Stallman and
Roland McGrath.
Copyright (C) 1988, 89, 90, 91, 92, 93, 94, 95, 96,
97
Free Software Foundation, Inc.
This is free software; see the source for copying
conditions.
There is NO warranty; not even for MERCHANTABILITY
or FITNESS FOR A
PARTICULAR PURPOSE.
Report bugs to <bug-gnu-utils@prep.ai.mit.edu>.
```

トラブル②　C++のプログラムをコンパイルできない

・ソースファイル名の拡張子をC++の拡張子にしてください。C++の拡張子は.cpp、.cxx、.ccなどですが、デフォルトでC++のソースファイルとしてコンパイラが認識する拡張子は、コンパイラの種類によって異なります。

・明示的にC++のコンパイラを使ってコンパイルしてください。たとえば、Linuxなどgccシステムでは**g++ progname.cpp -o progname**でコンパイルします。

コンパイル時のエラーメッセージとその対処

トラブル①　ヘッダファイルが見つからないと報告された

- ヘッダファイル名が間違っている可能性があります。タイプミスのほかに、システムやコンパイラによってヘッダファイル名が異なる場合や、新しいヘッダファイルがサポートされていない場合があるので、コンパイラのドキュメント（ヘルプやそのほかの情報）を参照してください。

- ヘッダファイルの検索パスを正しく指定してください。ほとんどのコンパイラで、ヘッダファイルの検索パスはオプション**-I**で指定できます。LinuxのようなUNIX系OSでは、**ln**コマンドを使ってヘッダファイルのあるディレクトリにリンクを設定することができます。次の例は、ソフトリンクで/usr/src/linux3-3/include/linuxを/usr/include/linuxに設定する例です。

```
# cd /usr/include
# ln -s /usr/src/linux3-3/include/linux linux
```

- サポートされていないヘッダファイルを使っている場合は、以前のヘッダファイルに変更してください。たとえば、<locale>がサポートされていないコンパイラでは<locale>の代わりに<locale.h>を使い、ソースコードもそれに合わせて変更する必要があります。

トラブル②　usingやnamespaceが定義されていない識別子として報告された

- コンパイラが名前空間（ネームスペース）をサポートしていない場合、`using namespace std;`の行は削除する必要があります。その場合、名前の競合や`::`の解決が行えない場合があり、そのようなときにはソースプログラムを変更する必要があります。

トラブル③　関数が定義されていない識別子として報告された

- コンパイラが名前空間（ネームスペース）をサポートしている場合、`using namespace std;`を忘れていないかチェックしてください（ほとんどの場合、`std`名前空間を使います）。また、そのほかの必要な名前空間がある場合には`using namespace xxx;`（`xxx`は名前空間名）を追加してください。

B-02　コンパイル時のエラーメッセージとその対処

- 自分で作った関数が定義されていないと報告される場合は、プロトタイプ宣言を行います。自分で作った関数を、定義より前に参照している場合、関数のプロトタイプを宣言する必要があります。関数のプロトタイプ宣言とは、関数の名前、戻り値、引数の型を宣言することです。プロトタイプ宣言はdouble myfunc(int argn);のように最後にセミコロン（;）が必要ですから注意してください。

- 必要なライブラリをリンクするように指定しないと、定義されていないか、未解決のシンボルがあるというメッセージが表示されることがあります。UNIX環境の標準的なC/C++コンパイラの場合、たとえば、math.hをインクルードする関数を使うプログラムでは、次のようにコンパイルオプション**-lm**を付ける必要があります。

```
cc -lm -o myprog main.c
```

トラブル④　演算子が定義されていないと報告された

- 必要なヘッダファイルのインクルードを忘れると、このエラーが発生します。たとえば、C++のソースプログラムのstringの入出力で<<や>>を使うときには#include <string>を追加してください。

トラブル⑤　cinやcoutなどが定義されていない識別子として報告された

- 必要なヘッダファイルのインクルードを忘れるとこのエラーが発生します。#include <iostream>を追加してください。

- 名前空間をサポートしている場合、using namespace std;を忘れるとこのエラーが発生します。

- 明示的にC++のコンパイラを使ってコンパイルしてください。たとえば、Linuxなどgccシステムでは**g++ progname.cpp -o progname**でコンパイルします。

トラブル⑥　特定のライブラリの関数が未定義の識別子として報告された

- 必要なライブラリをリンクしてください。たとえば、スレッド関連の関数が未定義の識別子として報告されるときには、マルチスレッド対応のライブラリをリンクしてください。Visual C++の場合、［プロジェクトの設定］ダイアログボックスの［C/C++］ページの［コード生成］で［使用する標準ライブラリ］に［マルチスレッド］を選択します。

トラブル⑦ プリコンパイル済みヘッダの検索中に予期しないEOFを検出したと報告された

- プリコンパイルするものとして指定したヘッダファイルが、ソースファイルの中で正しく使われていません。#include文を調べて、プリコンパイルするヘッダファイルの名前を正しく指定するか、プリコンパイル済みヘッダを使わないようにします。
- 変更された個所に関連するファイルだけをコンパイルするように設定している場合、依存関係の解決が完全に行われず、その結果、プリコンパイルが完全に行われないことがあります。そのような場合には、すべてのソースファイルを強制的に完全に再コンパイル（再ビルド）しなおすと問題が解決することがあります。

トラブル⑧ 特定の文字が認識できないと報告された

- ソースファイルが壊れている可能性があります。
- コメントやリテラル文字列以外の場所で2バイト文字の空白や句読点（、。）、その他の日本語文字などを使っている可能性があります。

プログラミング豆知識

メモリリーク

メモリを割り当てて使用したら、使い終わったあとで解放しなければなりません。メモリブロックを、C言語の関数calloc()、malloc()、realloc()などを呼び出して割り当てたときにはfree()を呼び出して解放します。C++のnew演算子を使ってメモリを割り当てたときにはdelete演算子を使って解放します。OSとライブラリのメモリ管理機能がメモリリークを防ぐために完全に機能すれば、万一、メモリの解放を忘れても、プログラムが終了したときにそのプロセスが使っていたメモリは自動的に解放されます。しかし、たとえそうであっても、メモリは原則としてプログラマの責任で解放しなければなりません。たとえば、長期間連続して実行されるサーバーのプログラムなどでは、メモリリークが蓄積してメモリ不足の状況に陥ることがあります。プログラムの種類によってはメモリリークはとても重大な問題になるので、ソースコードを解析してメモリリークを検出するための専用のツールもあります。

実行時のトラブルとその対処

トラブル①　実行可能ファイルが見つからないというエラーメッセージが表示される

・コンパイルは成功したが実行可能ファイルが見つからないというエラーメッセージが表示される場合には、環境変数PATHに「./」が含まれていない可能性があります。ファイルがカレントディレクトリにあることを示すために**./hello**のように実行可能ファイル名の前に「./」を付けるか、**/hello/hello**または**/home/myname/hello/hello**のように、実行可能ファイルを絶対パスで指定します。

トラブル②　何も表示されない

・開発環境によっては、コンソールプログラムを実行すると、ウィンドウが表示されてその中で実行され、プログラムが終了すると共にそのウィンドウが閉じてしまうことがあります。そのような場合にはプログラムのmain()の最後、あるいはそのほかの適切な場所に次のような行を追加して、ウィンドウが自動的に閉じないようにしてください。

```
printf("Hit any key");
getch();
```

トラブル③　表示される文字がおかしい

・漢字コードが異なるシステムでは、ソースファイルを適切な文字コードに変換しなければなりません。UNIX環境では漢字コードの変換にコマンド**iconv**を使うことができます。たとえば、シフトJISのソースプログラムをUTF-8に変換するときには、次のようにします。

```
iconv -f sjis -t utf-8 src.c > dest.c
```

また、nkfを使ってシフトJISのソースプログラムをEUCに変換するときには次のようにします。

```
nkf -e file.sjis > file.euc
```

- UNIX環境では、エスケープシーケンスなどのバックスラッシュを使うところで、円記号（¥）ではなくバックスラッシュ（\）を使う必要があります。
- バッファリングを行う出力関数や演算子を使うときには、クラスの異なるものを使うと、予期しない結果になることがあります。たとえば、<stdio.h>と<iostream.h>および<iostream>を同時に使うと、同期をとらない限り、期待したとおりに出力される保証はありません。

トラブル④　Segmentation faultでコアダンプされる

- 不正なメモリ領域にアクセスしようとした可能性があります。割り当てていないメモリを使おうとしたり、変数の大きさより大きいデータを変数に保存していないか、すでに解放したメモリを使っていないか調べてください。同じポインタに対してfree()を2回以上呼び出していないかということも調べてください。

トラブル⑤　デバッグビルドでは正常に機能するが、リリースビルドでは機能しない

- リリースビルドのコンパイルオプションをチェックしてみてください。assert()のようなデバッグ用関数やマクロは、通常、リリースモードでは評価されません。したがって、アサート文の中に記述した式や関数の呼び出しは実行されません。たとえば、Visual C++でリリースモードを指定すると自動的にNDEBUGが定義されますが、NDEBUGが定義されているときには、assert()のようなデバッグ用関数は評価されません。また、関数によってはデバッグとリリースでは動作が異なるものがあります。

環境の違いが原因となるトラブルと対策

トラブル①　ファイルにアクセスできない

- Windowsはパスのデリミタとして、スラッシュ（/）と円記号（¥）の両方を使うことができます。しかし、UNIXでは、パス名やファイル名を表す文字列でデリミタとしてスラッシュしか使えません。
- 大文字/小文字が違っている可能性があります。UNIXでは大文字/小文字を区別します。WindowsのFATファイルシステムでは大文字小文字を区別しません。Windows NT/2000/XPでインストール可能なNTFSでは、ディレクトリ一覧については大文字小文字を区別しますが、ファイル検索などのシステム操作では大文字小文字を無視します。
- ファイルの属性をチェックしてください。ファイルやディレクトリへのアクセス権がなければアクセスできません。

プログラミング豆知識

フォルダとディレクトリ

　Macintoshと最近のWindowsでは、ディレクトリのことをフォルダといいます。フォルダはファイルやほかのフォルダを保存するための比較的抽象性の高い論理的な場所を指し、ディレクトリはソフトウェア的にどこかに存在する場所を指します。また、シンボリックリンク（UNIX系OS）、フォルダのショートカット（Windows）とフォルダのエイリアス（Macintosh）なども含めると厳密にはフォルダとディレクトリは完全に同じとはいえません。しかし、一般ユーザーにとってはフォルダとディレクトリは同じものを別の言葉で表したものとみなされる傾向があります。ファイルシステムの操作などのプログラミングでは、通常、フォルダではなくディレクトリという言葉を使います。

付録C：プログラミング用語集

ここではC/C++プログラミングの用語のうち、本書で使っている特に重要な用語を説明します。

C-01　プログラミング用語集

プログラミング用語集

言葉は、さまざまな意味を持つことがあり、同じ言葉を状況によって異なる意味で使うことがあります。特にコミュニティーによって使われる用語はかなり異なります。たとえば、UNIX系OSの開発者とWindowsの開発者では、同じ意味のことがらに異なる表現を使うことがよくあります。

ここでは、C/C++を使ったプログラミングで最も一般的と思われる意味や表記を優先して示すように努めましたが、場合によっては、ここにない意味を持つことや表記が異なること、違う意味で使われることなどがあります。

また、厳密な定義はともすると抽象的になりがちなので、C/C++でプログラムを作成するという観点から、この用語集ではわかりやすさを優先して説明しています。

アルファベット

ANSI

The American National Standards Institute（米規格協会）の略。ANSIは商業や通信に関連するさまざまな規格を規定する組織です。ANSIが規定したC言語の規格をANSI C、ANSIが規定したC++をANSI C++といいます。ANSIキャラクタセットはANSIが規定した文字セットのことです。ANSI標準に従うことで、異なるコンピュータシステムや異なる実行環境でプログラムやデータを移植したり利用できます。ANSI Cで規定されているC言語の関数をANSI関数といいます。

API

Application Programming Interface（アプリケーションプログラミングインターフェイス）の略。アプリケーションやアプリケーションが直接使うプログラムコンポーネントが、ファイルシステムへのアクセスやGUIへの出力のような低水準サービスを必要とするときに使うインターフェイスのこと。たとえば、Win32APIはWindowsのプログラムが、ウィンドウやメニュー、ダイアログボックスなどを作成したり操作するときなどに使うインターフェイスです。

ASCII文字列

ASCII文字セットで構成された文字列。ASCII文字セット（ASCII character set）は標準英語キーボードのキーに対応する文字や記号を表わす7ビット（0から127までの128個）のコード体系で、英語の文字、数字などを主に使うコンピュータで幅広く使われている文字

セットです。ASCIIは、American StandardCode for Information Interchange（米国情報交換用標準コード）の略です。
参照→ISO646

EOF

End Of Fileの略。ファイルの最後を表します。

GUIプログラム

Graphical User Interface（グラフィカルユーザーインターフェイス）を使うプログラム。GUIプログラムでは、表示にグラフィックスを多用して情報を基本的にウィンドウに表示し、マウスのようなポインティングデバイスを使って操作します。情報の入力に主にキーボードを使い、文字だけを表示する従来の文字ベースのユーザーインターフェイス（CUI）と区別するときに使います。

IDE

Integrated Development Environment（統合開発環境）の略。プロジェクト単位あるいはワークスペース単位でソースコードファイルを編集し、プログラムをコンパイル（ビルド）できるソフトウェア開発用ソフトウェアのこと。代表的なIDEの例として、Visual C++やC++ Builderなどがあります。通常、IDEは、開発のためのさまざまなツールを備えていますが、さらにその環境内でデバッグできるようにもなっているので、IDDE（Integrated Development Debug Environment、統合開発デバッグ環境）と呼ぶこともあります。

ISO646

ISO646は文字集合の規格で、いわゆるASCII文字について定めています。ISOはInternational Organization for Standardization（国際標準化機構）の略です。C++プログラミングでは、`and`（&&）、`and_eq`（&=）、`bitand`（&）、`bitor`（|）、`compl`（?）、`not`（!）、`not_eq`（!=）、`or`（||）、`or_eq`（|=）、`xor`（^）、`xor_eq`（^=）などの演算子または区切り子を定義している標準ヘッダファイルiso646.hが特に重要で、このヘッダの定義はC言語でも使用可能です。

POSIX

Portable Operating System for UNIXの略。オープンシステム環境標準を規定したIEEE標準。UNIXとプログラムの間の言語インターフェイス、セキュリティ、ネットワーク、リアルタイム処理などを定義しています。POSIXに準拠したプログラムは、理論的にはPOSIXをサポートするすべてのマシンで実行できます。POSIXはUNIXシステムサービスを基にして標準が規定されていますが、Windows NT/2000/XPのサブシステムPOSIXでも実行できます。

C-01 プログラミング用語集

STL

Standard Template Libraryの略。C++言語のANSI標準テンプレートライブラリで、リストやキュー、スタックのようなコンテナと、コンテナの内容を操作するプログラムであるアルゴリズムが定義されています。

Unicode関数

引数や戻り値の受け渡しにUnicode文字またはUnicode文字列を使う関数。引数や戻り値の受け渡しにANSI文字またはANSI文字列を使うANSI関数に対して使います。

UNIX系OS

Linux、BSD、FreeBSD、Solaris、Mac OSなどの、UNIXから生まれたり、UNIXと互換性のあるOSの総称。

X Window System

UNIXで事実上の標準となっているウィンドウシステム。Xウィンドウシステム、X Windowシステム、X-Window、あるいは、単にXともいいます。X Window Systemはサーバープログラム(Xサーバー)とクライアントプログラム(Xクライアント、Xアプリケーション)から構成され、X Window Systemのクライアントプログラムは、X Toolkit(Xt)やMotifのようなGUIライブラリを使って作成します。

Xlib

X Window Systemのクライアントプログラムを作るときに使う最も基本的なライブラリ。Xサーバーとの通信、最も基本的なリソース(グラフィック、フォントなど)の管理、イベントの処理に必要な基本的な作業などを行います。

あ行

アサーション(assertion)

検証とレポートの機構。デバッグのために、式を評価して、その結果が`false`である場合に診断メッセージを表示してプログラムを停止すること。たとえば、値が0であるとあとのコードで重大な問題が発生する可能性があるような場合、その前に`assert(p != 0);`のようなアサーションのためのコードを記述します。するとpが0のときに、“Assertion Failed”のようなメッセージが、ファイル名とその行位置情報と共に出力されます。通常、アサーションはデバッグビルドのときだけ有効です。

あと入れ先出し（LIFO）

最後に保存されたデータが最初に取り除かれること。たとえば、A、B、Cという3個のデータが保存されたデータは、C、B、Aという順に取り出されます。LIFOは、Last In First Outの略です。スタックは一種のLIFOのデータ保存領域です。
参照→先入れ先出し（FIFO）

アドレス（address）

メモリあるいはオブジェクトやドキュメント上の位置。メモリアドレスは、メモリ空間の特定の位置を指す言葉です。C/C++プログラミングではメモリアドレスのことを単にアドレスと呼ぶことがあり、また、実際の物理的なメモリ上の位置ではなく、ディスク上のスワップスペースやメモリ上での動的な移動なども考慮に入れた論理的な位置と考えることもあります。

アラインメント（alignment）

メモリ上やウィンドウの中など、ほかのオブジェクトの中における、あるオブジェクトの整列方法。たとえば、メモリに4バイトの値を保存するときに、任意のアドレスから4バイトの長さに保存することもできますが、先頭から2の倍数または4の倍数の位置に保存すると、より合理的にアクセスできることがあります。このような配置の方法を決定することをアラインメントの指定といいます。また、構造体のメンバの配置ではアラインメントを1にするとメモリ上の空白がなくなり、使用メモリ量を減らすことができます。

アルゴリズム（algorithm）

特定の問題を解決するための処理や計算の手順、あるいは計算方法。C++のプログラミングでは、STLコンテナの内容を操作するための関数群を特にアルゴリズムと呼びます。

アロケータ（allocator）

メモリやレジスタなどの割り当てを行うもののこと。メモリアロケータは、特定のオブジェクトにメモリを割り当てる手段を提供します。

インクリメント（increment）

値を増やすこと。プログラミングの世界では、単にインクリメントするという場合、通常、値を1だけ増加させることを指します。

インクルード（include）

外部のものを取り入れること。ファイルをインクルードするということは、ファイルをそこに挿入することで、インクルードファイルは挿入されるファイルのことです。

C-01　プログラミング用語集

インスタンス（instance）
同じ種類の物が存在できる可能性があるもので、そのときに実際に存在する1つ。たとえば、あるクラスからは複数のオブジェクトを作成できますが、そのひとつひとつがインスタンスです。

インターフェイス（interface）
広義には、ハードウェアとハードウェア、ソフトウェアとソフトウェア、あるいはシステムと人間の間などで、接点となるものや接点となるプログラムすべて。システムと人間の接点であるヒューマンインターフェイスから、プログラム間の接点となるCOMのインターフェイスまで、さまざまなインターフェイスがあります。
オブジェクト指向プログラミングでは、関数を宣言しただけで、コードを定義していないものを指します。

インタープリタ言語（interpreter language）
インタープリタと呼ばれる、ソースコードまたは中間コードプログラムを実行するプログラムで翻訳（解釈）されて実行されるプログラミング言語。代表的なインタープリタ言語には、Visual BasicやJavaがあります。

インデント（indent）
字下げのこと。ソースコードで、読みやすくするために論理的な部分ごとに一定の字下げすることをインデントするといいます。たとえば、関数やforループの{　}の中のコードを行の前に空白を入れて右にずらして記述するのがインデントです。

インプリメンテーション（implementation）
実際に実行されるコードのこと。実装ともいいます。たとえば、関数の名前や型、引数などを宣言する部分に対して、実際に実行される関数の定義のコード部分をインプリメンテーションといいます。

エスケープシーケンス（escape sequence）
表示したり印刷できない文字を表現したり、通常の処理とは異なる方法で解釈するときに使う文字の連結。たとえば、C言語で出力の改行に使う¥nは、¥とnという2個の文字として出力されるのではなく、1つのつながったもの（シーケンス）として出力され、改行として処理されます。

エラーコード

関数がエラーになったときに返される値。一般的にはエラーコードはグローバル変数errnoにセットされます。

演算子（operator）

算術計算や比較など（演算）に使う記号。たとえば「足し算の記号」は加算演算子で、C/C++で大小を比較するときに使う>や<は比較演算子です。

エントリー関数

C言語のmain()関数のように、そのプログラムまたはモジュールが実行されるときに最初に呼び出される関数。エントリーポイントともいいます。

オーバーフロー（overflow）

あふれること。たとえば、変数に保存可能な値より大きな値なったり、あらかじめ確保してあるメモリ領域を越えてほかの用途に使っているメモリに書き込んだときに、オーバーフローしたといいます。値を0にきわめて近い小さな値で割るような演算の結果としてオーバーフローが発生することもあるので注意する必要があります。

オーバーライド（override）

オブジェクト指向プログラミングで、基本クラスにあるメソッドと同じシグネチャのメソッドを派生クラスで定義してそれを呼び出すようにすること。

オーバーロード（overload）

同一のスコープ内で同一の関数や演算子に複数の定義を与えること。オーバーロードが可能な場合、たとえば、DoSomething ("買い物")とDoSomething("買い物", "散髪")などのように、同じ名前の複数の関数を定義できます。実行時に実際に呼び出される関数は、呼び出すときの関数の引数の数と型に引数が一致する関数です。

オブジェクト指向プログラミング

再利用可能なプログラムコードを蓄積することを主な目的として、関連あるデータとコードをひとまとめにして、1つのオブジェクトとして扱うプログラミング手法。C++ではデータを保存するメンバ変数と、コードを実装するメンバ関数からなるクラスを使ってオブジェクト指向プログラミングを行います。

C-01　プログラミング用語集

か行

カウンタ（counter）

ループの繰り返し回数やオブジェクトが参照された回数などを数えるための変数またはオブジェクト。

型（type）

コンピュータで扱うデータの形式。データ型ともいいます。コンピュータでデータとして扱う数値や文章のような情報には、その性質上、同じようには扱えないものがあります。たとえば、名前と金額を加算しても意味がありませんし、実際に意味のある演算はできません。また、アルファベットの文字1文字のように1バイトで表現できるデータと、長い文字列のように数バイトかそれ以上の長さの記憶領域（メモリ）を使わないと表現できないデータがあり、このようなデータを無秩序に混在させるとメモリや演算の効率が悪くなります。そこで、このような性質が異なるデータをいくつかの型に分類して扱います。構造体はユーザーが定義した一種の型とみなすこともできるので、構造体（や、ときにはクラス）を型と呼ぶこともあります。また、void以外の関数は値を返すので、返す値の型を関数の型とみなします。

型宣言

データ型を明示的に指定するために行う型の宣言。

環境変数（environment variabble）

OSの実行環境を規定するための変数。環境変数の種類や名前、表現と設定方法などはOSや使用するアプリケーションによって違いますが、一般的によく使われる環境変数には、たとえば、次のような環境変数があります。

PATH：　実行可能ファイルを探すディレクトリを表します。
　　　　（例：PATH=C:¥Windows;C:¥bin）
LANG：　言語環境を表します。（例：LANG=ja_JP.sjis）

記憶クラス

オブジェクトや変数の存在期間（寿命）、リンクの方法やデータの操作方法などを分類するもの。C/C++の記憶クラスには、auto、register、static、externなどがあります。

キャスト（cast）

型を強制的に変更すること。たとえば、(int) v;は値vの型を強制的にintにします。

共用体（union）
サイズや型が異なるオブジェクトを保存できる一種のデータ構造。たとえば、メンバがcharとintである共用体に1つの値を保存して、あるときにはcharとして、またあるときにはintとしてその値（または値の一部）を取り出すことができます。

空白（white space）
スペースまたはタブのこと。より広い意味では、改行や復帰なども含みます。

クラス（class）
特定の種類のオブジェクトのプロパティやメソッドを定義したもの。クラスを一種のひながた（テンプレート）としてオブジェクトを作成します。

クラスライブラリ（class library）
関連ある一連のC++のクラスを定義したライブラリ。たとえば、MFC（Microsoft Foundation Class）ライブラリは、Windowsアプリケーションのフレームワークを提供する一連のクラスを集めたものです。

グローバルスコープ（global scope）
モジュール、プロシージャ、メソッドなどの一定の範囲を超えた、より大きな範囲のこと。たとえば、関数の外で定義した変数はそのモジュールのどの変数からでもアクセスできるので、グローバル変数といいます。

構造体（structure）
1個以上のメンバ（構成要素）からなるデータ構造。C言語やC++ではstructを使って宣言します。ユーザー定義のデータ型ともいいます。

コメント（comment）
注釈のこと。C言語では/*と*/ではさまれた文字列、C++では//以降行末までがデフォルトのコメントです。

コレクション（collection）
何らかの関連があるオブジェクトを集めたオブジェクト。

コンテナ（container）
ほかのオブジェクトを入れることができるオブジェクトのこと。STLのコンテナクラスは、ほかのデータ型のデータやほかの種類のオブジェクトを保存できるオブジェクトを作るためのクラスです。

C-01　プログラミング用語集

コンパイル（compile）

C/C++言語のような人間が読むことができる高水準プログラミング言語で記述されたソースコードプログラムや、テキスト形式（リッチテキスト形式やHTML形式を含む）で記述されたドキュメントなどを、コンパイラと呼ぶ専用のソフトウェアを使って機械語やその他の形式のファイルに翻訳すること。

コンポーネント（component）

構成要素のこと。たとえば、大きなアプリケーションは、多数のより小さなソフトウェアコンポーネントを使って作ります。

さ行

先入れ先出し（FIFO）

最初に保存されたデータが最初に取り除かれること。たとえば、A、B、Cという3個のデータが保存されたデータは、A、B、Cの順で取り出されます。FIFOは、First In First Outの略です。キューはFIFOバッファの一種です。
参照→あと入れ先出し（LIFO）

サブシステム

主システムには含まれない主システムに従属するシステム。Windows NTやWindows 2000/XPではPOSIXはサブシステムとして実装されています。

シーケンス（sequence）

データやそのほかのオブジェクトが連続していること。たとえば、文字列を、文字を構成するバイトデータが連続しているものとみなしたときに、バイトのシーケンスであるといいます。シーケンスの中のシーケンスをサブシーケンスといいます。

ジェネレータ（generator）

値やそのほかのオブジェクトを生成するもの。たとえば、乱数ジェネレータは乱数を生成します。

識別子（identifier）

プログラムで使う、変数、型、関数、ラベルなどに付ける名前。シンボルともいいます。

シグナル（signal）
実行中のプログラムに送られる信号。普通は、プログラムの通常の流れとは違う割り込まれた処理を行うために発生させる信号、あるいはそのための概念を指します。

シグネチャ（signature）
関数の名前、戻り値、引数からなる関数の宣言。

指数値
数式で、累乗を表わすために数字や文字の右上に表示する小さな数字の値。

静的（static）
プログラムの実行中に柔軟に変更されないこと、または、メモリから削除されないで値を持ちつづけること。たとえば、静的リンクはプログラムを実行する前にあらかじめリンクすることです。静的変数は、プロシージャが再度呼び出されたときに値が再初期化されない変数です。
参照→動的

システムコール（system call）
OSがサポートする低レベルの機能を呼び出すこと。たとえば、ディレクトリやファイルなどを作成したり削除するようなOSがサポートしている基本的なルーチンを呼び出すこと。そのルーチンそのものを指すこともあります。

実装
参照→インプリメンテーション

条件式（conditional expression）
結果が真か偽になる式。通常、条件判断に使いますが、結果を変数に代入することもあります。

初期化式（initializer expression）
forステートメントで、初期化の1つとして評価される式。

処理系
ソースドキュメントを処理して結果を生成するソフトウェアのこと。ソースプログラムをコンパイルして目的のファイルを生成するコンパイラも処理系に含まれますが、広い意味では、コンパイラに付属しているライブラリやそのほかの基本ツールなども含めることがあります。

C-01　プログラミング用語集

シンタックス（syntax）

プログラムの構文。シンタックスエラーは、プログラムの構文の間違いのこと。

シンボリックデバッグ（symbolic debug）

デバッグ用のシンボルを埋め込んだ実行可能なファイルを使って行うデバッグ。プログラムのコンパイル時にシンボリックデバッグ情報（単にデバッグ情報ともいいます）を埋め込むように設定してEXEやDLLなどを作成すると、シンボリックデバッグをサポートするデバッガを使ってそのプログラムを実行しながらデバッグすることができます。

シンボル（symbol）

プログラムで使う、変数、型、関数、ラベルなどに付ける名前。

スコープ（scope）

プロシージャや変数などの名前の有効範囲。
参照→グローバルスコープ

ステートメント（statement）

プログラム言語が直接サポートする命令。forステートメントは繰り返しの命令、ifステートメントは条件に応じて実行するコードを変える命令です。

ストリーム（stream）

データあるいは情報の流れのこと。入出力ストリームは、プログラムが外部のデバイス（ファイル、キーボード、ディスプレイ、プリンタ、通信ポートなど）に入出力するときの抽象的なデバイスです。

増分式

forステートメントや、ループの中でカウンタとして使う値を増やす式の俗称。一般的にはi++やj += 2;のように単純な式にします。

添字（index）

配列やリストの特定の要素を指定するための文字。たとえば、int v[100];の100や、n = x[y];のyのような数字または変数。添字を使って配列要素またはポインタを参照する式を添字式といいます。

た行

定数（constant）
プログラムの実行中に変わらない値。

ディレクティブ（directive）
コンパイラなどに処理方法を指示する命令。たとえば、#defineや#ifのように、コードがコンパイルされる前に評価される命令や、（コンパイラごとに個別にサポートしている）#pragmaディレクティブのようにコンパイラに特別な指示を行う命令を指します。指令ともいいます。

デクリメント（decrement）
値を減少させること。プログラミングの世界では、単にデクリメントするという場合、通常、値を1だけ減少させることを指します。

デスクリプタ（descriptor）
オープンしているファイルやデータベース、そのほかのオブジェクトを識別または説明するための値またはオブジェクト。データ構造あるいは構造体であるのが普通ですが、多くの場合、その内容を直接扱うことより、それを使ってデータやオブジェクトを参照することを主な目的とします。記述子、ディスクリプタともいいます。

デストラクタ（destructor）
C++で、コンストラクタが割り当てたすべてのリソースを破棄するために呼び出す特別なメソッドのこと。

手続き型プログラミング
コンピュータが行うべきことを一連の手続きとして記述するプログラミング方法。たとえば、文字列を出力し、値を入力し、計算して、結果を出力するというように、プログラムコード全体であらかじめ決まった1つの手続きを処理するようなプログラミング方法です。これに対して、ユーザーの操作やシステムの状況に応じて、そのとき適切なプログラムコードを実行するプログラムのプログラミング方法を、イベント駆動型プログラミングといいます。

デバイスドライバ（device driver）
デバイスを制御するためのプログラム。OSとハードウェアの間に位置し、個々のハードウェアの細かな違いを吸収します。

C-01　プログラミング用語集

デバッグ（debug）

プログラムのバグ（論理的な誤り）を修正すること。デバッグのための情報を含んだ特別な実行可能ファイルを生成することをデバッグビルド、そのようにして生成された実行可能ファイルをデバッグバージョンの実行可能ファイルといいます。

デリミタ（delimiter）

情報の区切り記号。実際にデリミタとして使われる記号（文字）は条件によって異なりますが、C/C++言語では、一般的にいって、スペース、改行、カンマ（,）、ピリオド（.）、セミコロン（;）、コロン（:）などがよく使われます。

テンプレート（template）

文書やプログラムなどを作るときのひな形。C++言語ではクラスや関数のテンプレートを定義しておくと、必要なデータ型に対応するクラスや関数の実際のコードをコンパイル時に生成することができます。C++のテンプレートを利用すると、あるクラスをさまざまなデータ型に適用できるので、データ型ごとにクラスを作成する必要がなくなります。なお、以前のC++コンパイラの中にはテンプレートをサポートしていないものもあります。

テンプレートクラス（template class）

ユーザーが指定した型引数に応じて、コンパイル時に適切な型のクラスとして展開できるクラスです。
参照→テンプレート

トークン（token）

コンパイラやインタープリタがソースプログラムやソースドキュメントを解析する際に最小の単位として認識する要素。たとえば、ifのようなキーワード、変数や関数などの名前、定数、演算子など。

な行

名前空間（namespace）

識別子を区別する空間（範囲）。名前空間をサポートしている環境では、異なる名前空間に同じ名前の識別子を定義して、それぞれ別のものとして識別できます。ネームスペース、ネーム空間ということもあります。

ネスト（nest）

ステートメントや関数、コメントなどを、同じ種類の別のステートメントや関数、コメントなどの中に入れ子状態で埋め込むこと。たとえばforステートメントのループの中にforステートメントを記述したり、C言語のコメント/* */の中に同じC言語のコメント/* */を記述することをネストといいます。ネストが可能な場合と不可能な場合が厳密に定義されていることが多いので注意が必要です。

は行

バイナリサーチ（binary search）

データの検索方法の1つで、検索対象を2つの部分に分けてそのどちらかを選ぶという手続きを繰り返すことで、求めるデータを検索する方法のこと。２分探索ともいいます。

パイプ（pipe）

プログラムの出力を別のプログラムの入力に接続するもの。OSのシェルの一般的なパイプ機能は、cmda | cmdbのように|を使ってあるプログラムの標準出力を別のプログラムの標準入力に送ります。

配列（array）

同じ種類の複数の要素の並び。配列の各要素には順序があり、それぞれの要素はインデックス番号で識別します。たとえば、変数配列は同じデータ型で同じ名前の複数の要素からなるもので、個々の要素はインデックス（添字）で識別します。

バッファ（buffer）

データを一時的に保存する場所。プログラムの入力や出力をいったん保存しておく変数やメモリ領域のことを指すことがよくあります。たとえば、通信アプリケーションのようなプログラムでは、受信したデータを常に即座に処理できる保証はないので、受信したデータはバッファに一時的に保存しておき、処理できるときにバッファの中のデータを順番に処理します。

パディング（padding）

空いた場所を埋める詰めもののこと、または詰め物をすること。たとえば、数値を出力するときに、値の桁が何桁であっても先頭に0（ゼロ）を追加して00123のような形式ですべて5桁で出力したいときに使うゼロのことをパディング文字といいます。

C-01　プログラミング用語集

パラメータ（parameter）

C/C++プログラミングでは、主に、関数の引数やプログラムの起動時に指定する引数のこと。広義には、なんらかの操作を行うときに、対象や母集団の特性を示したり、限定したり制限するための値。パラメーターと表記することもあります。

反復子（iterator）

C++でオブジェクトを保存してあるコンテナの内容にアクセスするときに使うオブジェクト。特定のアドレスにあるデータにアクセスするときに使うポインタに似ています。

ヒープ（heap）

プログラムで利用できるメモリ領域の1つ。プログラマが任意に確保して変数などに割り当てて使うことができるメモリ領域のことで、比較的大きなデータを保存するときに使います。C言語プログラムでは、関数malloc()とfree()を使って、C++ではキーワードnewとdeleteを使ってヒープのメモリを割り当てたり解放します。

ヒープメモリ（heap memory）

ヒープとして使うことができるメモリ領域。変数やオブジェクトに割り当てられるヒープメモリの量は、システムの構造やメモリ（実メモリと仮想メモリ）の量によって決まります。

引数（argument）

関数に渡すパラメータ。たとえば、func(int v, char c)の場合、vやcが引数です。

ビット（bit）

コンピュータで扱うデータの最小の単位。C/C++はビット演算子を使ってビット単位で演算や評価を比較的高速で行うことができます。

ビットセット（bit set）

ビットの集合。たとえば、2進数表示で0101は4ビットのビットセットです。

標準入出力

C/C++プログラムで標準で自動的に開かれる入出力デバイスのこと。通常、デフォルトで、標準入力はキーボード、標準出力と標準エラー出力はディスプレイですが、デバイスを変更したり、リダイレクトやパイプで入出力先のデバイスを指定することもできます。

符号なし（unsigned）

符号ビットがないこと。符号付きの数では、データの特定のビットを符号（+/-）の情報を伴う値として扱いますが、符号なしの数は符号として解釈するビットがありません。

復帰（carriage return）

印字や出力の位置を左に戻す動作。一方、改行は印字や出力の位置を下に移動します。ただし、OSや言語、標準ライブラリによって改行と復帰の解釈にはさまざまなパターンがあり、改行に改行と復帰の意味を含めたり、復帰に改行と復帰の意味を含めることがあります。一般的には改行に復帰の意味も含めますが、厳密には狭義の改行だけでは垂直方向の位置が次の行に移動するだけで、左には戻りません。

浮動小数点数（floating point number）

1.23や456.852などの小数点以下の数がある数を表すときに使う値。実数の表現方法の1つです。1.23は値が123で小数点位置が1桁目のあと、456.852は456852で小数点位置が3桁目のあと、というように、小数点を除いた数字の並びと、小数点の位置を保存するので、大きな値と小さな値を同じ方法で保存することができます。C/C++ の浮動小数点数型は、`float`、`double`、`long double`です。

フラグ（flag）

プログラムの実行状況やデータの属性などを示す一種のマーカー。たとえば、`fDirty`というような名前の変数を宣言して、ドキュメントが更新されたときに`true`にし、更新されたドキュメントが保存されたら`fDirty`を`false`にすることで、更新された情報を確実に保存できるようにすることがありますが、このような変数をダーティーフラグまたは更新フラグといいます。

フラッシュ（flush）

データをバッファから押し出すこと。たとえば、ディスクにデータを書き込むときには、個々のデータはバッファに保存され、一定の量がたまると実際にディスクに書き込まれますが、その動作を強制的に行うことをフラッシュするといいます。

プリプロセッサ（preprocessor）

主作業の前処理として行う作業を行うもの。通常、プログラムのコンパイルの前処理としてテキストを処理するものを指します。プリプロセッサは、通常、ソーステキストの解析は行わず、インクルードファイルをインクルードしたり、マクロを展開したりします。

C-01　プログラミング用語集

プロジェクト（project）

広義には、特定の作業全体のこと。プログラミングでは、主としてプログラムを開発するために必要なもの全体を指します。たとえば、「プロジェクトのファイル」という場合、あるアプリケーションのためのプロジェクトファイルやプログラムファイルに加えて、データや関連ドキュメントファイルなども含まれます。

プロセス（process）

実行中のプログラム（通常はアプリケーションプログラム）のインスタンス。たとえば、あるアプリケーションを起動すると、インスタンスが1つ作成され、それが1つのプロセスになります。プロセスは、それぞれ仮想アドレス空間、プログラムコード、データを所有し、プログラムの実行に必要なリソースを確保します。プロセスの中でさらに作成したプロセスを子プロセスといいます。

プロセッサ時間

CPUが処理のために割り当てる時間。

ブロックスコープ

大かっこ{　}で限定した範囲。C++ではコードの中で{　}を使ってその{　}の中だけで有効な変数や関数を宣言することができます。

プロトタイプ宣言

C++ではすべてのシンボルはあらかじめ宣言しておかないと使うことができません。関数を定義するときには、あらかじめ関数の型、引数の型と名前（識別子）だけを宣言する（実行するコードは記述しない）ことができ、このような宣言をプロトタイプ宣言といいます。

ヘッダコメント

モジュールの先頭のコメント。通常、そのモジュールの機能の概要や作成者などをコメントとして記述します。

変数（variable）

プログラムの実行中に変更できる特定の型のデータを入れる名前付きの場所。通常、適切な名前を付けて宣言し、数値、文字列、そのほかのデータを代入したり、演算に使います。
参照→定数

ポインタ（pointer）

指し示すもの。通常、メモリ上の値を指す値を単にポインタといい、ディスクファイル上の現在のアクセス位置を指すものをファイルポインタと呼びます。また、マウスの現在の位置を指している画面上のオブジェクト（通常は矢印型）のことをマウスポインタといいます。メモリ上の値を指すポインタはメモリ上のアドレスと考えても構いませんが、必ずしも実アドレスを指しているわけではありません。

ポインタ宣言

ポインタの変数を宣言すること、またはその宣言そのもの。
参照→ポインタ

補数（complement）

各桁の数を基数-1から引いた数。たとえば2進数の0010（10進数で2）の1の補数は1111-0010=1101です。

ホワイトスペース（white space）

空白とみなされる、スペース、タブ、改行、復帰などの総称。狭い意味ではスペースとタブのこと。
参照→空白

ま行

マクロ（macro）

C/C++ではプリプロセッサがプリプロセス（前処理）で置き換える文字列。広い意味では特定の目的で行われる一連の操作を記録保存したもの。

マニピュレータ（manipulator）

C++でオブジェクトの状態を操作するオブジェクト。たとえば、入出力ストリームのマニピュレータdec、oct、hexは、数値をそれぞれ10進数、8進数、16進数で出力するように操作し、endlは改行します。

マルチバイト（multibyte）

2バイト以上のオブジェクト。たとえば、マルチバイト文字は2バイト以上で1文字を表す文字のことです。

C-01　プログラミング用語集

メソッド（method）

オブジェクトに属しているひとまとまりのプログラム部分。メンバ関数ともいいます。メソッドはサブルーチンであるという点で関数やプロシージャ、ステートメントなどに似ていますが、特定のオブジェクトに作用するという点で、ほかのものとは違います。

メンバ（member）

構造体やクラスの中で定義されている変数や関数のこと。広義では構成要素のこと。メンバーということもよくあります。

メンバ関数（member function）

クラスの要素として定義されている関数のこと。メンバー関数、メソッドともいいます。

モジュール（module）

独立して独自の機能を果たすように設計されたもの。C/C++のプログラミングでは、特に1つのソースファイルを1つのコンパイル単位と考えて、それぞれをモジュールとみなします。たとえば、main.cとmisc.cという2個のソースファイルからなるプログラムは、2個のモジュールから構成されているプログラムです。
広義では、さまざまな実行可能ファイル、DLL、デバイスドライバなどから、「拡張モジュール」などと呼ばれることがある特定の機能を持ったハードウェアまでを指します。

モジュロ（modulo）

整数を整数で割った余り、または余りを求める計算。

文字列定数（string literal）

プログラムの中で変更されない文字列。リテラル文字列ともいいます。

戻り値（return value）

関数が返す値。リターン値、返り値、戻し値、返し値、あるいは返値などと呼ぶこともあります。

や・ら行

ユーザー定義（user defined）

プログラミングで「ユーザー定義」というときは、プログラマが定義したことを意味します。たとえば、「ユーザー定義の関数」とは、プログラマが作った関数のことです。

ライブラリ（library）
関数やクラスなどをまとめたファイル。一連の関連がある関数やクラスを1つのライブラリにまとめることで、関数やクラスが利用しやすくなります。

ライブラリ関数（library function）
特定のライブラリに含まれる関数。たとえば、C言語の標準ライブラリに含まれる関数のことを標準ライブラリ関数といいます。

ランタイムライブラリ
実行時に使うライブラリ。特に、特定のプログラミング言語に標準で備わっていて、そのプログラミング言語のプログラムを実行する際に必要不可欠なライブラリのことを指します。Windows系のコンパイラ（IDE）の場合、標準Cライブラリを含む言語に付属するライブラリのことをランタイムライブラリと呼ぶことがよくあります。

リソース（resource）
広い意味では、CPU、データ、メモリなど、プログラムの実行時に資源とみなせるあらゆるもの。プログラミングでは、特に、プログラムで使うメニュー、アイコン、文字列のようなプログラムが使うデータを指すことがあります。

リダイレクト（redirect）
出力先をデフォルトとは異なる別の場所に変えること。たとえば、**cmd > file**とすると、コマンド**cmd**の標準出力への出力がファイルfileに書き込まれます。

リテラル文字列（string literal）
文字列定数として使う文字列のこと。

リリース（release）
実際にエンドユーザーが使えるようにすること、あるいは、デバッグ情報を含まないモジュールのことやそれを生成すること。後者は通常は、デバッグの反対の意味で使います。
参照→デバッグ

リンク（link）
C/C++ のプログラミングでは、複数のソースコードをコンパイルして作成した複数のオブジェクトモジュールやライブラリを結合して1つの実行可能ファイルを作成すること。
広義では、連結すること、あるいは連結したもの相互を結びつけるもののこと。

C-01　プログラミング用語集

ループ式

forステートメントで繰り返しのたびに実行される式。

例外（exception）

プログラムの実行中に発生する、プログラムの実行に重大な影響を与えるできごとのこと。例外処理は、プログラムの実行中に発生した例外に対処するための処理です。

レジスタ（register）

CPUの中にある、演算あるいは比較などの特定の目的のためにデータを保存する領域。

列挙型（enumerated type）

enumステートメントを使って定義した一連の名前付定数。1つのenumで定義した一連の定数を、新しいデータ型として使うこともできます。

ロード（load）

メモリにオブジェクト（コードやデータなど）を読み込むこと。

ロケール（locale）

システムやプログラムを実行するときの、国（あるいは文化圏）別の環境設定。表示文字列の言語、文字列のソート順、キーボードの配置、日付と時刻の書式などがロケールに影響されます。

論理演算（logical operation）

ビットごとの和や積、比較、抽出など、算術演算以外の演算。

付録D：参考リソース

本書で参照している書籍や、C/C++プログラミングに関する知識をさらに深めたり、UNIX系OSやWindowsでプログラミングを始めるために役立つ書籍やWebページを紹介します。

D-01　参考リソース

参考リソース

参考書籍（日本語）

①プログラミング言語C（K&R）

Brian Kernighan、Dennis Ritchie著　石田晴訳（共立出版）、ISBN4-320-02692-6

②Linuxプログラマーズ辞典

リチャード・ピーターセン著トップスタジオ監訳（翔泳社）、ISBN4-88135-780-8

③GNU C++プログラミング

トム・スワン著（翔泳社）、ISBN4-88135-956-8

④独りで習うC

日向俊二 著（翔泳社）、ISBN4-7981-0899-5

⑤コンピュータサイエンス入門

日向俊二 著（カットシステム）、ISBN4-87783-122-3

参考資料および書籍（英語）

①ANSI X3J16/ISO WG21 Joint C++ Committee Working Paper for Draft Proposed International Standard for Information Systems-- Programming Language C++

②The C++ Programming Language Third Edition Bjarne Stroustrup, Addison-Wesley, ISBN 0-201-88954-4

Webページ

①C/C++

http://www.att.com/attlabs/

②GTKとGTK+

http://www.gtk.org/

③Gtk--

http://gtkmm.sourceforge.net/

④V

http://www.objectcentral.com

⑤Microsoftのプログラミング情報

http://msdn.microsoft.com/library/ja

http://support.microsoft.com/

索引

2章、3章、4章のリファレンスの項目をアルファベット順に並べた索引と、本文中の用語や関数および2章、3章、4章のリファレンス項目を機能から逆引きするための索引を用意しました。

目的別INDEX

リファレンス索引

INDEX

INDEX

逆引き用索引

INDEX

INDEX

●マ行

INDEX

●ヤ行

●ラ行

●ワ行

用語INDEX

INDEX

INDEX

INDEX

●さ行

INDEX

●は行

INDEX

●ま行

●著者紹介

日向俊二
コンピュータサイエンティスト、ソフトウェアエンジニア。前世紀の中ごろにこの世に出現し、FORTRANやC、BASICでプログラミングを始め、その後、主にプログラミング言語とプログラミング分野での著作、翻訳、監修などを精力的に行う。わかりやすい解説が好評で、現在までに、コンピュータサイエンス、C/C++、Java、Visual Basic、C#、XML、アセンブラなどに関する著作多数。
E-Mail:shunji_hyuga@infoseek.jp

クリエイティブ・ディレクション　佐藤直樹（アジール・デザイン）
アート・ディレクション&デザイン　溝端 貢（アジール・デザイン）
DTP　西村満枝

C/C++辞典　第3版
関数・シンタックス・アルゴリズム　逆引きリファレンス

2000年5月31日　初版第1刷発行
2002年8月9日　第2版第1刷発行
2006年10月10日　第3版第1刷発行

著　者　日向 俊二（ひゅうが しゅんじ）
発行人　佐々木 幹夫
発行所　株式会社 翔泳社（http://www.shoeisha.co.jp）
印刷・製本　大日本印刷株式会社

ISBN4-7981-1201-1　Printed in Japan

ISBN4-7981-1201-1

C3055 ¥2800E

9784798112015

株式会社 翔泳社

定価：本体 2,800円+税

1923055028005

C++はC言語を拡張して作成された言語です。C言語の知識にC++の知識を追加し、オブジェクト指向の考え方を理解すれば、C言語とC++の両方を自在に使いこなせるようになります。 (「はじめに」より)